

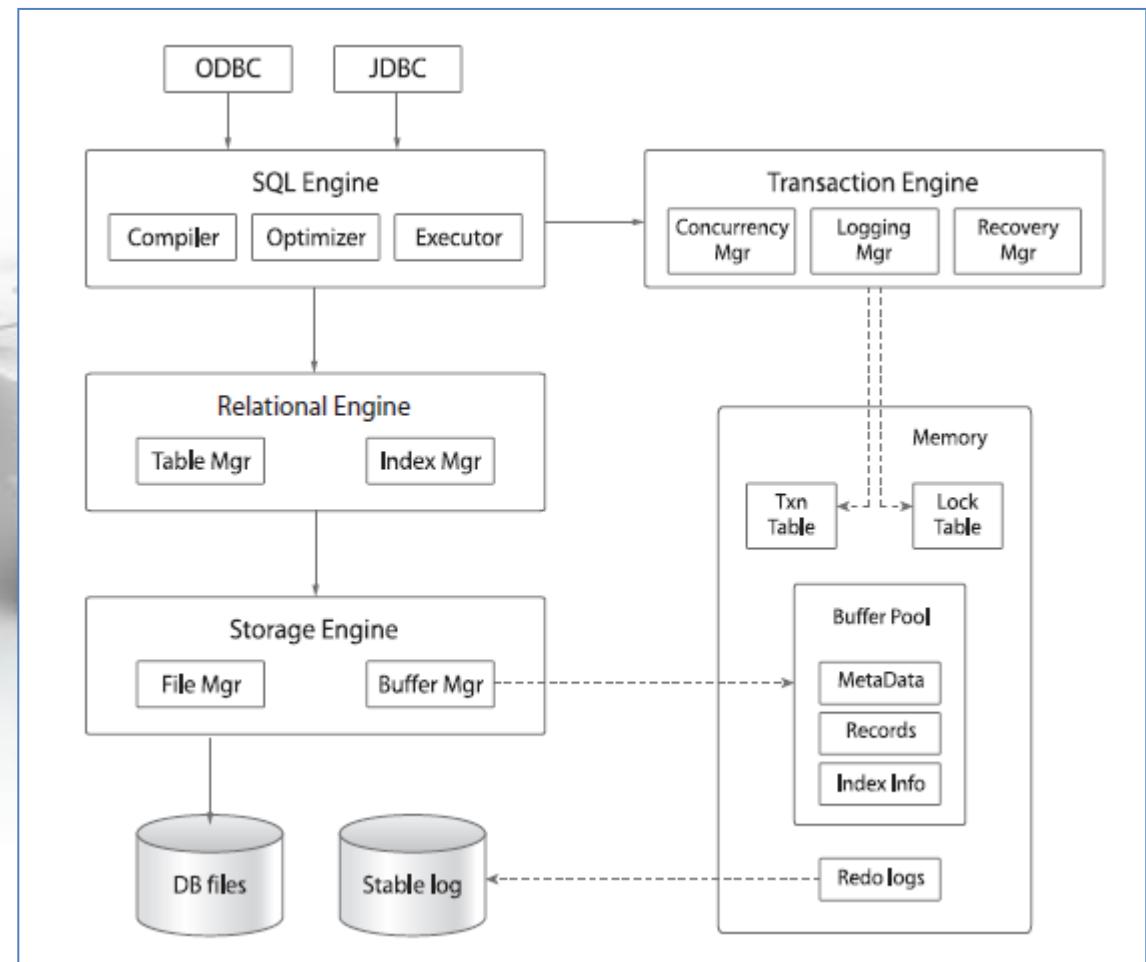
Intro

데이터베이스 기본

데이터베이스란?

- 데이터베이스의 정의와 개념 (<http://ko.wikipedia.org>)
 - "데이터베이스(database)는 여러 응용 시스템들의 통합된 정보들을 저장하여 운영할 수 있는 공용 데이터들의 묶음이다.
여러 사람이 공유하고 사용할 목적으로 통합 관리되는 정보의 집합이다.
논리적으로 연관된 하나 이상의 자료의 모음으로 그 내용을 고도로 구조화함으로써 검색과 간접적 효율화를 꾀한 것이다.
즉, 몇 개의 자료 파일을 조직적으로 통합하여 자료 항목의 중복을 없애고 자료를 구조화하여 기억시켜 놓은 자료의 집합체라고 할 수 있다."

데이터베이스

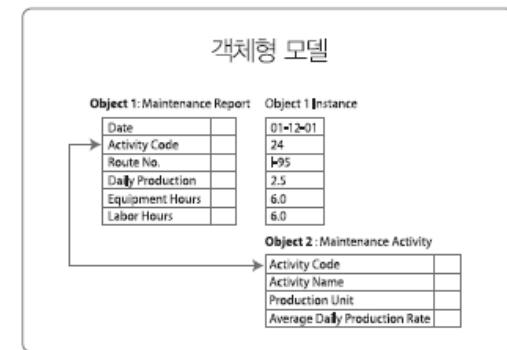
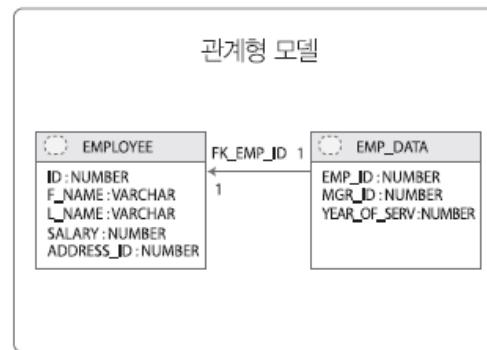
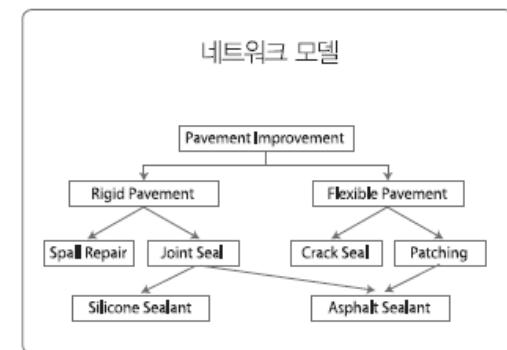
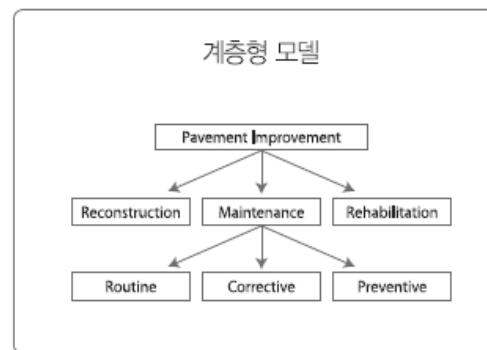


데이터베이스 장점

- 데이터 중복 제거
- 권한이 없는 접근 제한(보안과 권한 시스템 구축 필요)
- 지속적인 저장 공간과 질의 처리 공간 제공
- 백업과 복구 지원
- 여러 사용자 인터페이스 제공
- 데이터 사이에 다양한 관계를 효율적으로 표현
- 데이터 **무결성** 보장
- 규칙을 사용하여 추론하고 수행
- 표준 정의
- 응용 프로그램 개발이 쉽고, 요구 사항이 변경될 때 쉽게 대처

데이터베이스 모델

- 다양한 데이터베이스 모델



데이터베이스 동향

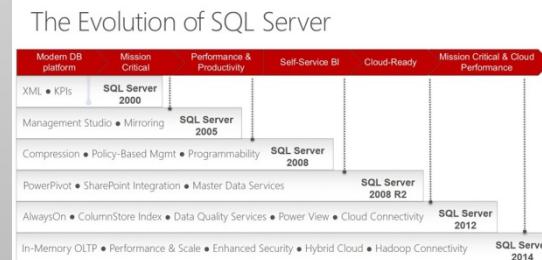
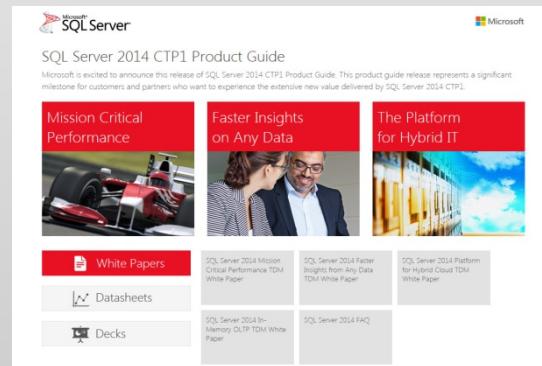
- Oracle

: New multitenant architecture that makes it easy to deploy and manage database clouds



- Microsoft

: SQL Server 2014 (w/ Azure)

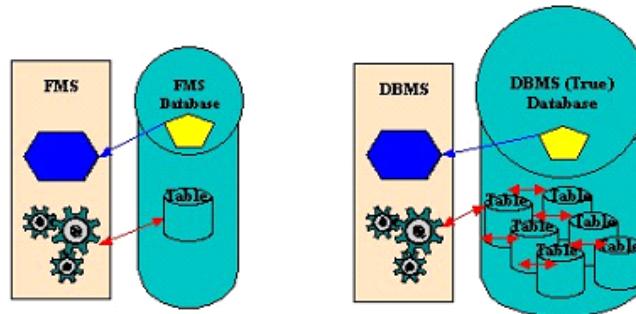


▪ NoSQL



파일 관리 vs. 데이터베이스

- 멀티유저
- 많은 기능
- 유연성



FMS versus DBMS Comparison Diagram (Figure 1)

File Management Systems

Advantages

Simpler to use

Less expensive

Fits the needs of many small businesses and home users

Popular FMS's are packaged along with the operating systems of personal computers (i.e. Microsoft Cardfile and Microsoft Works)

Good for database solutions for hand held devices such as Palm Pilot

Disadvantages

Typically does not support multi-user access

Limited to smaller databases

Limited functionality (i.e. no support for complicated transactions, recovery, etc.)

Decentralization of data

※ Reference : <http://perwez.wordpress.com/2007/09/21/database-management-system-vs-file-management-system/>

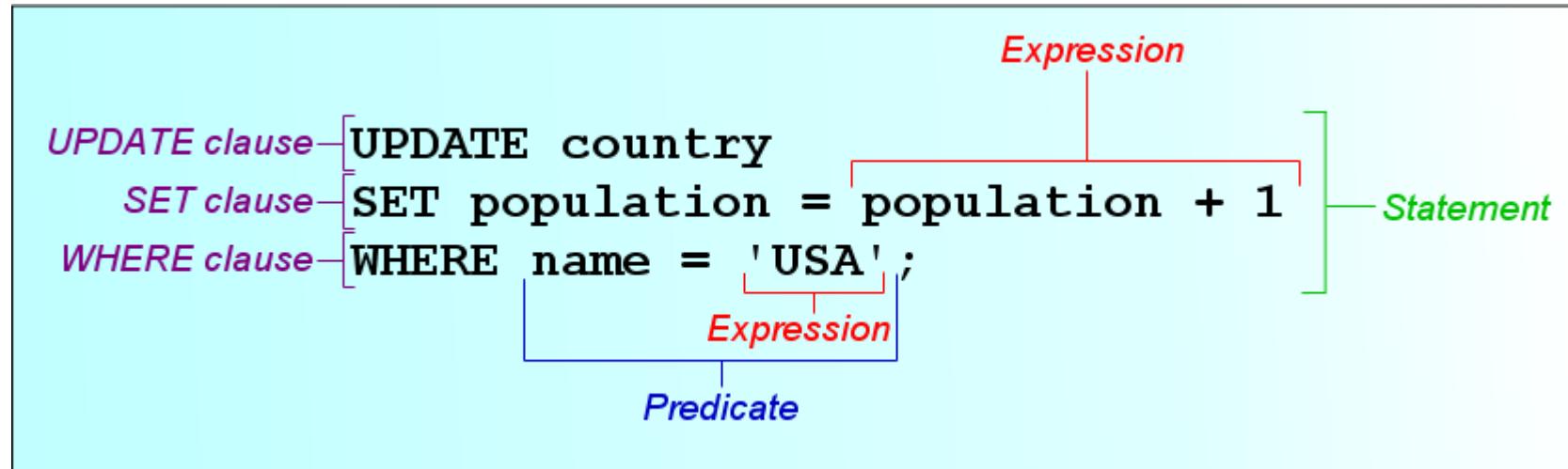
관계형 데이터베이스

- 정의 : 관계의 집합으로 구성된 데이터베이스
- 구성요소
 - 스키마 수정을 위한 인터페이스 드라이버
 - SQL 엔진
 - 트랜잭션 엔진, 저장엔진
- 트랜잭션 : ACID

구분	내용
원자성(Atomicity)	트랜잭션이 완료된 이후의 데이터베이스에는 데이터베이스 요소들에 대한 동작이 모두 수행되거나 아니면 하나도 수행되지 않아야 한다.
일관성(Consistency)	여러 가지 제약 사항들은 항상 일정하게 유지되어야 한다.
고립성(Isolation)	트랜잭션은 다른 트랜잭션이 동작하고 있지 않은 경우에만 수행되어야 한다.
내구성(Durability)	일단 트랜잭션이 완료되면 그 결과는 유실되어서는 안 된다.

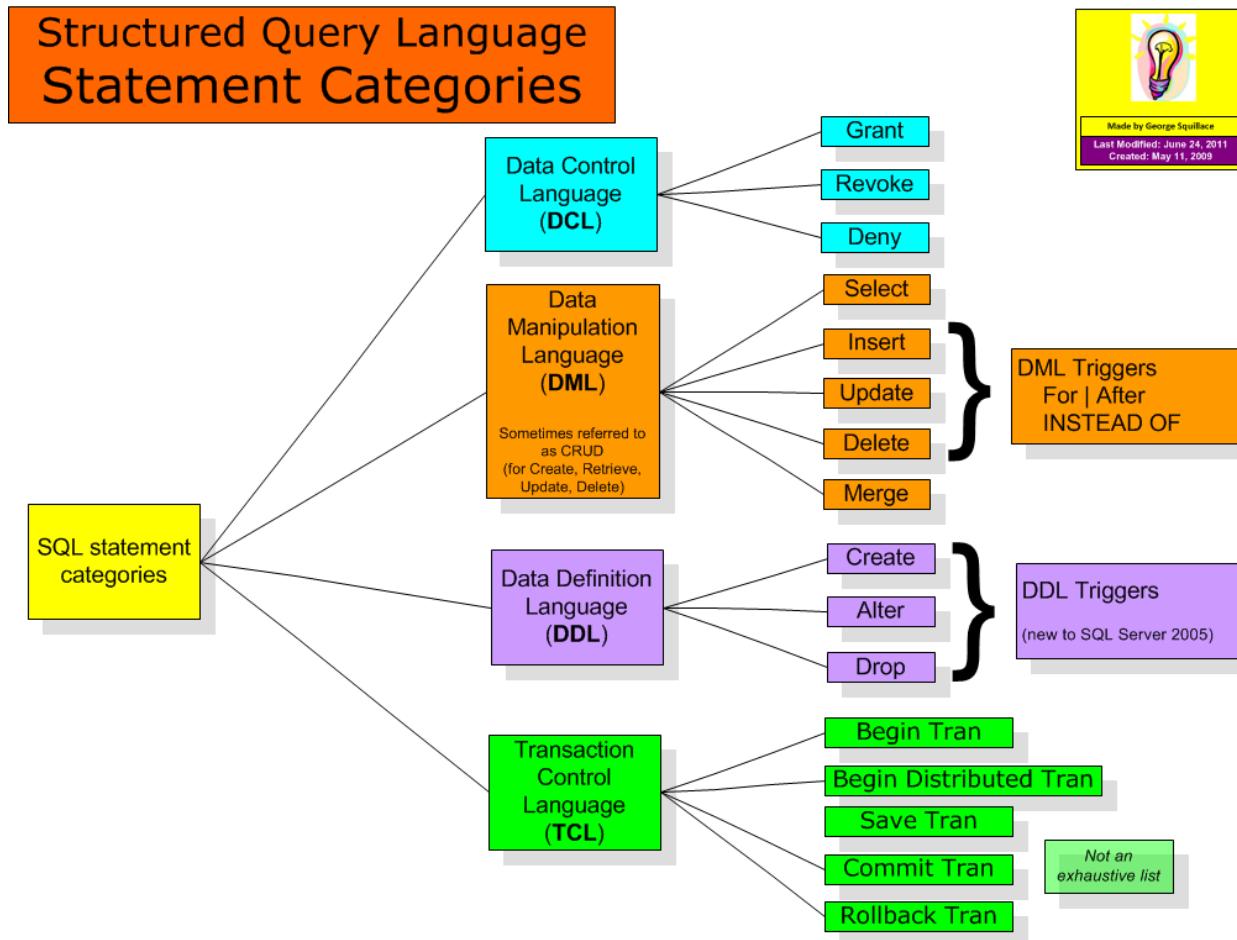
데이터베이스 언어

- SQL : Structured Query Language



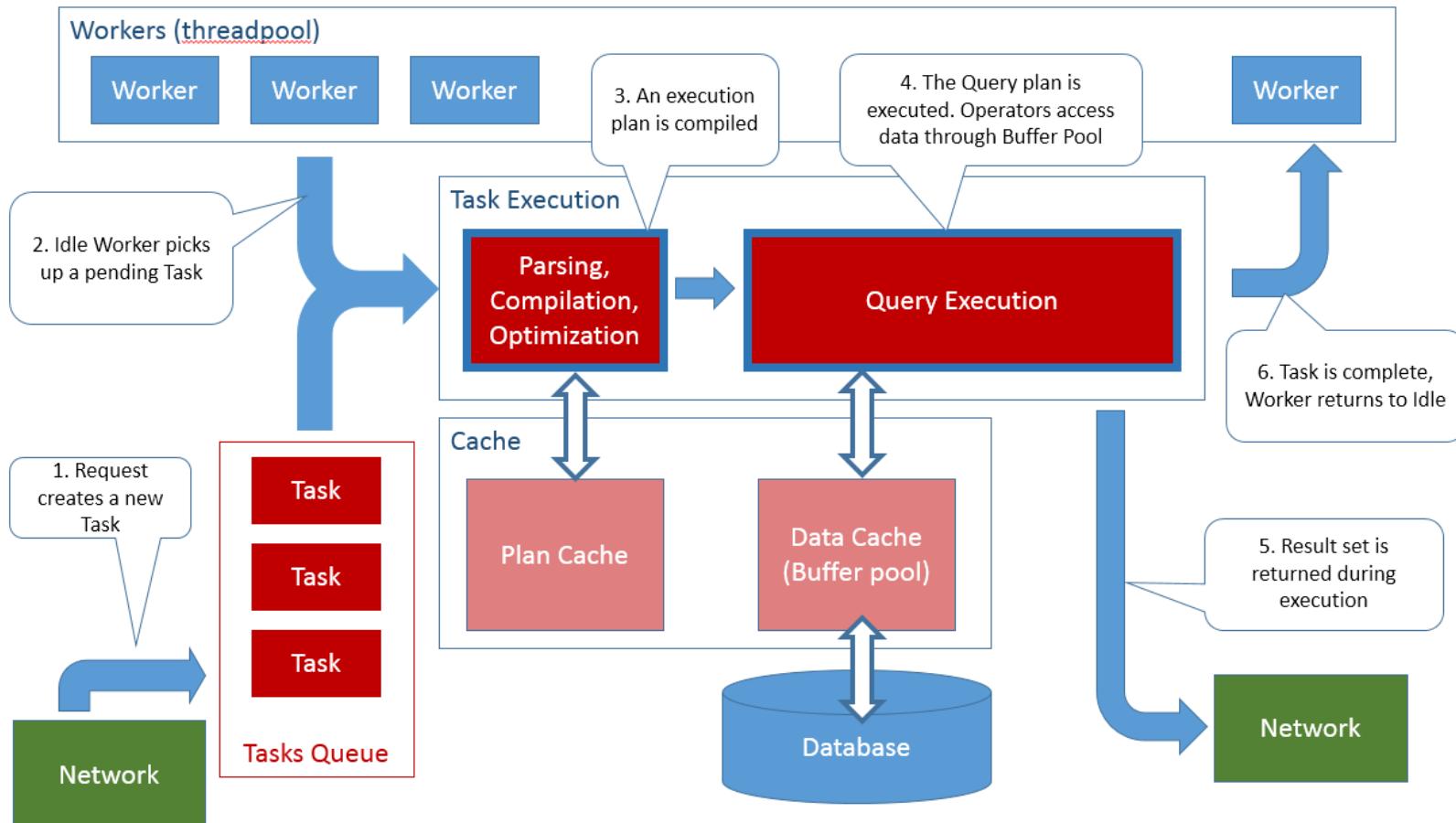
※ Reference : http://commons.wikimedia.org/wiki/File:Sql_statement_anatomy.png

SQL : Structured Query Language



※ Reference : <http://www.e-squillace.com/tech/techdiagrams/>

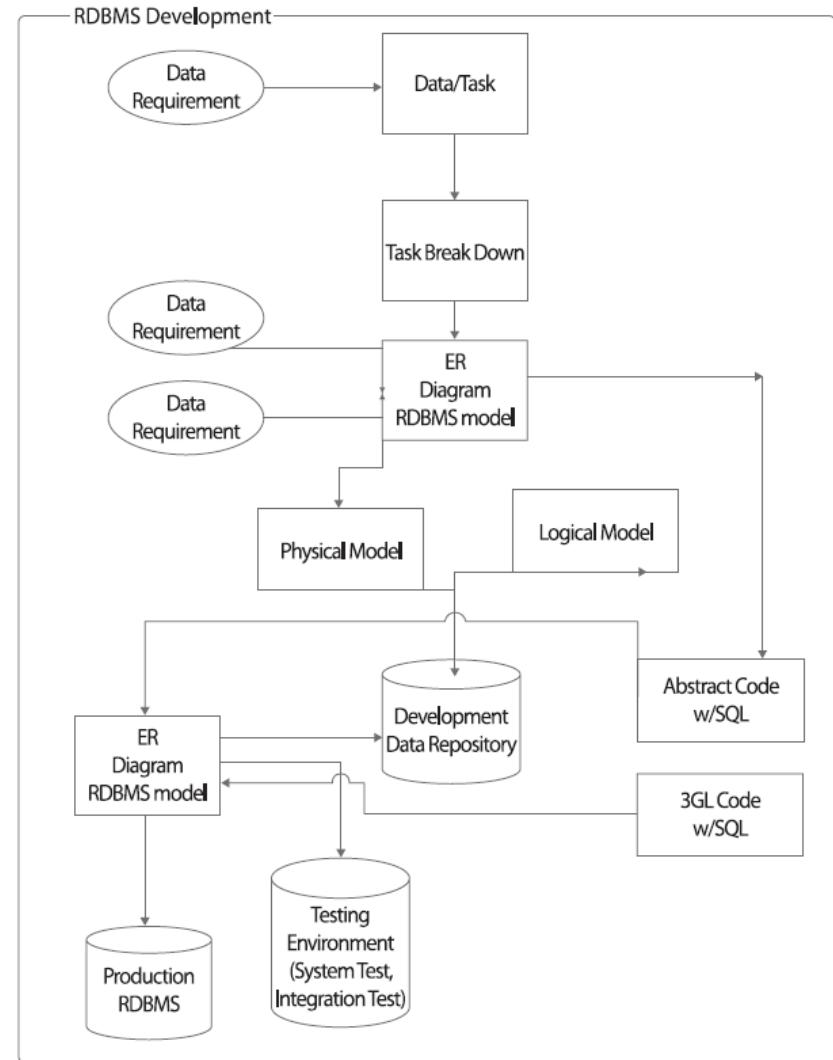
SQL Server operation



* Reference : <http://rusanu.com/2013/08/01/understanding-how-sql-server-executes->

How to develop DBMS?

- 용도에 맞는 데이터와 태스크 지정
- ER 다이어그램을 이용한 데이터 모델링
- SQL을 사용한 코드



객체지향형 데이터베이스

- 정의 : 객체 지향 데이터모델에 기반한 데이터베이스

구분	객체지향형 DBMS	객체관계형 DBMS
복합 객체 지원 여부	지원	지원
유일한 객체 식별자 지원 여부	지원	부분 지원(유일 키 없을 때)
캡슐화 지원 여부	지원	지원
클래스 구조 지원 여부	지원	지원
상속성 지원 여부	지원	지원
다형성 지원 여부	지원	미지원
확장성 지원 여부	지원	지원

NoSQL 종류와 특징

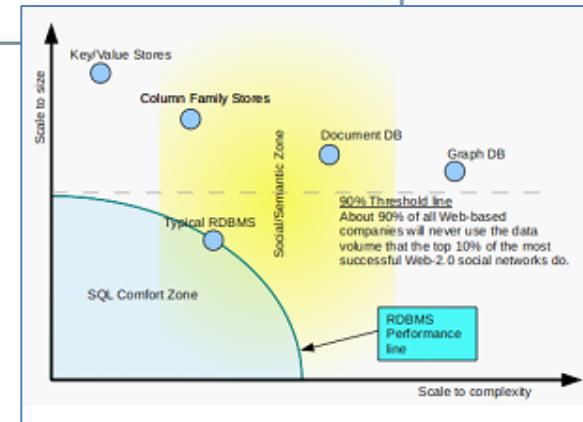
- 기본 내용 파악하기
- 다양한 NoSQL

Trend

Big Data Will Scale To Exabytes



※ Reference : http://www.syoncloud.com/big_data_technology_overview



NoSQL

Not
Only SQL

- 특징
 - 전통적인 관계형 데이터베이스 보다 덜 제한적
 - 일관성 모델을 이용하는 데이터의 저장 및 검색을 위한 매커니즘 제공
 - 디자인의 단순화, 수평적 확장성 추구
 - 단순 검색 및 추가 작업을 위한 매우 최적화된 키 값 저장 공간으로, Latency와 throughput과 관련하여 상당한 성능 이익 제공
 - 빅데이터와 실시간 웹 애플리케이션용

NoSQL

- Wikipedia에서의 NoSQL

<From Wikipedia, the free encyclopedia>

- A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.
- Motivations for this approach include simplicity of design, horizontal scaling and finer control over availability. NoSQL databases are often highly optimized key-value stores intended primarily for simple retrieval and appending operations, whereas an RDBMS is intended as a general purpose data store.
- There will thus be some operations where NoSQL is faster and some where an RDBMS is faster. NoSQL databases are finding significant and growing industry use in big data and real-time web applications.
- NoSQL systems are also referred to as "Not only SQL" to emphasize that they may in fact allow SQL-like query languages to be used.
- In the context of the CAP theorem, NoSQL stores often compromise consistency in favor of availability and partition tolerance.
- Barriers to the greater adoption of NoSQL data stores in practice include: the lack of full ACID transaction support, the use of low-level query languages, the lack of standardized interfaces, and the huge investments already made in SQL by enterprises.



Gigaom Research is a premium subscription site

[Sign up for a free 7 day trial](#)

CLOUD | REPORT

facebook, microsoft, google, +89 more

Report: NoSQL Databases – Providing Extreme Scale and Flexibility

by Matt Sarrel Jul. 6, 2010

This report underwritten by: Sarrel Group

 [Download the Full Report](#)

1 Executive Summary

As the data landscape changes, so must the databases used to gather, store and analyze the rich information within them. The emergence of web 2.0, social networking and user-contributed content have pushed traditional relational database management system (RDBMS) architectures to the limit. Consumer-facing Internet companies such as Google, Amazon, Yahoo! and LinkedIn are able to scale by using NoSQL (Not Only SQL)* data stores. Sites like these have realized that the controls of an RDBMS limit performance and lack the ability to conduct massive numbers of concurrent reads and writes. In cases where data must be stored and analyzed — and doesn't require the controls and rigidity of RDBMS — CIOs can learn from what's worked for hugely successful web sites, and therefore create architectures that promote flexibility and speed.

[Learn about our services](#) orContact us: [Email](#) / 800-906-8098

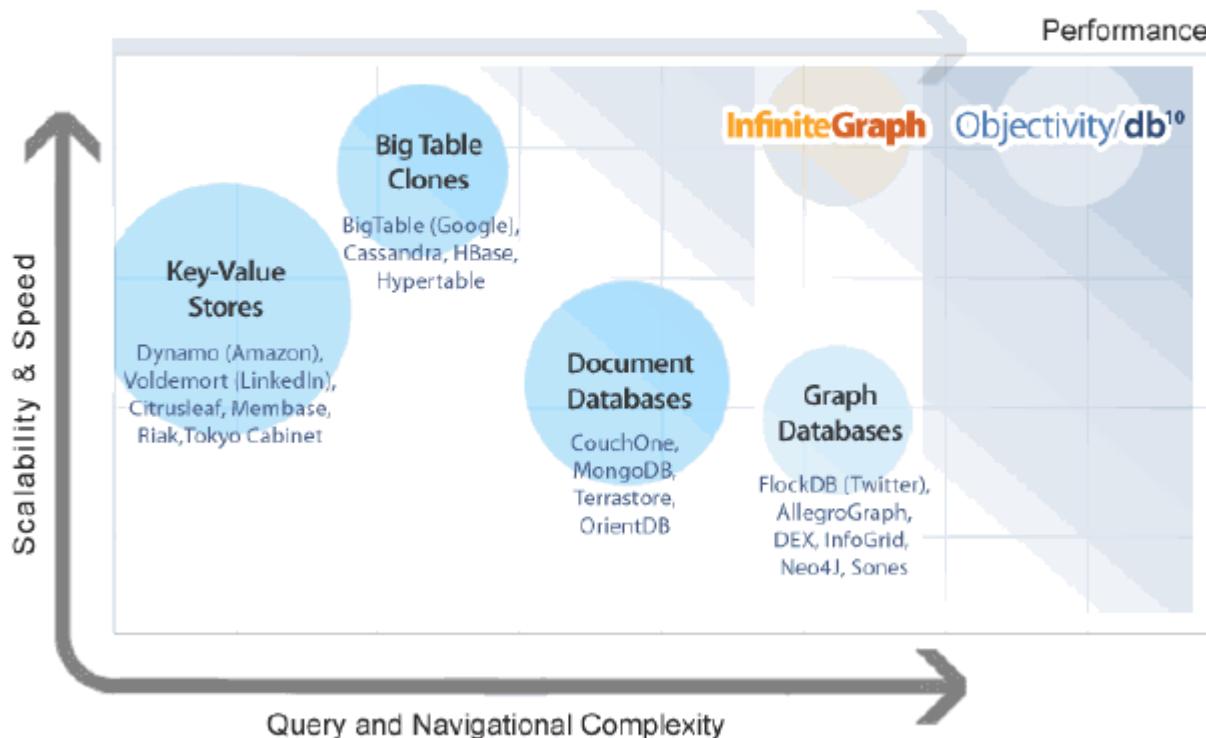
Table of Contents

- 1 Executive Summary**
- 2 Overview**
 - 2.1 Limitations of RDBMS Solutions**
- 3 A New Alternative Emerges: NoSQL and Unstructured Databases**
 - 4 NoSQL Market Overview**
 - 5 NoSQL Project Profiles**
 - 5.1 Key-value stores**
 - 5.2 Prominent Key-Value Store Project Analyses**
 - 5.3 Tabular or Columnar Data Structures**

※ Reference : http://research.gigaom.com/report/report-nosql-databases-providing-extreme-scale-and-flexibility/?utm_source=cloud&utm_medium=editorial&utm_content=dharrisstructure&utm_campaign=intext&utm_term=305182%20twitters-success-pulls-23-year-old-objectivity-into-nosql

NoSQL Solutions

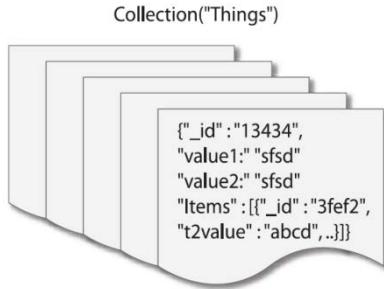
The "NOSQL" Technology Landscape



※ Reference : <http://gigaom.com/2011/03/04/twitters-success-pulls-23-year-old-objectivity-into-nosql/>

MongoDB

- 데이터를 JSON의 2진 버전인 BSON을 사용하여 문서에 데이터 저장



- 웹 응용 프로그램과 인터넷 기반 서비스를 위해 설계
- 읽기/쓰기 효율을 높임과 동시에 자동으로 장애조치를 수행하고 확장이 쉬움

The most popular ones

MongoDB (2.2)

- Written in: C++
- Main point: Retains some friendly properties of SQL. (Query, index)
- License: AGPL (Drivers: Apache)
- Protocol: Custom, binary (BSON)
- Master/slave replication (auto failover with replica sets)
- Sharding built-in
- Queries are javascript expressions
- Run arbitrary javascript functions server-side
- Better update-in-place than CouchDB
- Uses memory mapped files for data storage
- Performance over features
- Journaling (with -journal) is best turned on
- On 32bit systems, limited to ~2.5Gb
- An empty database takes up 192Mb
- GridFS to store big data + metadata (not actually an FS)
- Has geospatial indexing
- Data center aware

Best used: If you need dynamic queries. If you prefer to define indexes, not map/reduce functions. If you need good performance on a big DB. If you wanted CouchDB, but your data changes too much, filling up disks.

For example: For most things that you would do with MySQL or PostgreSQL, but having predefined columns really holds you back.

Riak (V1.2)

- Written in: Erlang & C, some JavaScript
- Main point: Fault tolerance
- License: Apache
- Protocol: HTTP/REST or custom binary
- Stores blobs
- Tunable trade-offs for distribution and replication
- Pre- and post-commit hooks in JavaScript or Erlang, for validation and security.
- Map/reduce in JavaScript or Erlang
- Links & link walking: use it as a graph database
- Secondary indices: but only one at once
- Large object support (Luwak)
- Comes in "open source" and "enterprise" editions
- Full-text search, indexing, querying with Riak Search
- In the process of migrating the storing backend from "Bitcask" to Google's "LevelDB"
- Masterless multi-site replication replication and SNMP monitoring are commercially licensed

Best used: If you want something Dynamo-like data storage, but no way you're gonna deal with the bloat and complexity. If you need very good single-site scalability, availability and fault-tolerance, but you're ready to pay for multi-site replication.

For example: Point-of-sales data collection. Factory control systems. Places where even seconds of downtime hurt. Could be used as a well-update-able web server.

Open Source Database Solution

- 소스코드 공개
- 제한없는 사용
- 라이선스 준수

Google search results for "opensource database". The results page shows various open-source database projects like PostgreSQL, MySQL, and Firebird.

검색 결과 약 125,000,000개 (0.34초)

관련 검색: [open database](#) [open source database](#) [open source database market share](#) [open source database monitoring tools](#) [open source database compare tool](#)

HOT 토픽

- 자산 1400억 주문발
- SBS 웃찾사 개그맨 기소
- 별그매 전지현 김수현
- 日趋아이 19금
- 아베 야스쿠니 참배 자체
- 설 연휴 2년차 국경구상
- 北 김정은 외조부 제주
- 지재를 주중 北 대사
- 달샤벳 수빈 스마일 오타
- 귀성길 일부 정체 절정

사이트

PostgreSQL: The world's most advanced open source database
www.postgresql.org/ 이 페이지 번역하기

Sophisticated **open-source** Object-Relational DBMS supporting almost all SQL constructs, including subselects, transactions, and user-defined types and ...
Downloads – Documentation – Development snapshots – About

CUBRID – Open Source Database Management System Optimized ...
www.cubrid.org/ 이 페이지 번역하기

Open Source RDBMS – Seamless, Scalable, Stable and Free On behalf of CUBRID **open source database** project I wanted to take a chance and make a ...

MySQL :: The world's most popular open source database
www.mysql.com/ 이 페이지 번역하기

"MySQL provides the perfect blend of an enterprise-level **database** and a cost-effective technology solution. In my opinion, MySQL is the only **database** we ...

Firebird: The true open source database for Windows, Linux, Mac ...
www.firebirdsql.org/ 이 페이지 번역하기

Join Firebird Join Firebird Foundation to support Firebird SQL development and receive multiple bonuses. Join! Follow Us. Select your media preference ...

The Open Source Database Benchmark
osdb.sourceforge.net/ 이 페이지 번역하기

A free, **open source**, performance benchmark for private testing of relational **database** systems. Tests are written in the C language using GNU tools.

RethinkDB: An open-source distributed database built with love
www.rethinkdb.com/ 이 페이지 번역하기

An **open-source** distributed **database** built with love. Enjoy an intuitive query language, automatically parallelized queries, and simple administration. Table joins ...

Document Database

Name	Language	Notes
BaseX	Java, XQuery	XML database
Cloudant	Erlang, Java, Scala, C	JSON store (online service)
Clusterpoint	C++	XML, geared for Full text search
Couchbase Server	Erlang, C, C++	Support for JSON and binary documents
Apache CouchDB	Erlang	JSON database
djondb[8][9][10]	C++	JSON, ACID Document Store
ElasticSearch	Java	JSON, Search engine
eXist	Java, XQuery	XML database
Jackrabbit	Java	Java Content Repository implementation
IBM Lotus Notes and Lotus Domino	LotusScript, Java, IBM X Pages, others	MultiValue
MarkLogic Server	XQuery, Java, REST	XML database with support for JSON, text, and binaries
MongoDB	C++, C#, Go	BSON store (binary format JSON)
Oracle NoSQL Database	Java, C	
OrientDB	Java	JSON, SQL support
CoreFoundation Property list	C, C++, Objective-C	JSON, XML, binary
Sedna	XQuery, C++	XML database
SimpleDB	Erlang	online service
TokuMX	C++, C#, Go	MongoDB with Fractal Tree indexing
OpenLink Virtuoso	C++, C#, Java, SPARQL	middleware and database engine hybrid

CouchDB, Redis



- CouchDB
 - 스키마가 필요없는 document 저장
 - JSON 포맷으로 데이터 교환
 - Erlang으로 개발
 - CouchDB 설치 시 웹서버가 같이 설치되어 Client와 HTTP로 통신하고 Data(Document)는 JSON으로 주고 받음
- 레디스(Redis)
 - 발전한 키-값 형태의 데이터베이스
 - 레디스에서 지원하는 키는 문자열과 해시(Hash), 리스트, 저장된 세트들
 - 문자열을 추가하거나, 해시에서 값을 증가시키거나, 리스트 추가와 세트 인터섹션(Intersection), 유니온(Union), 디퍼런스(Difference)에 대한 연산을 수행할 수 있고, 해당 동작에 대한 원자적 특징을 지원
 - 인-메모리 데이터 세트(In-Memory Dataset) 사용

CouchDB (V1.2)

- **Written in:** Erlang
- **Main point:** DB consistency, ease of use
- **License:** Apache
- **Protocol:** HTTP/REST
- Bi-directional (!) replication,
- continuous or ad-hoc,
- with conflict detection,
- thus, master-master replication. (!)
- MVCC - write operations do not block reads
- Previous versions of documents are available
- Crash-only (reliable) design
- Needs compacting from time to time
- Views: embedded map/reduce
- Formatting views: lists & shows
- Server-side document validation possible
- Authentication possible
- Real-time updates via '_changes' (!)
- Attachment handling
- thus, [CouchApps](#) (standalone js apps)

Best used: For accumulating, occasionally changing data, on which pre-defined queries are to be run.
Places where versioning is important.

For example: CRM, CMS systems. Master-master replication is an especially interesting feature, allowing easy multi-site deployments.

Redis (V2.8)

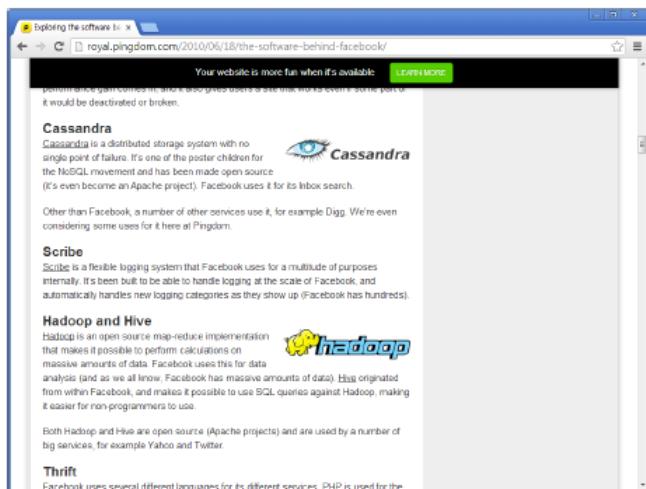
- **Written in:** C
- **Main point:** Blazing fast
- **License:** BSD
- **Protocol:** Telnet-like, binary safe
- Disk-backed in-memory database,
- Dataset size limited to computer RAM (but can span multiple machines' RAM with clustering)
- Master-slave replication, automatic failover
- Simple values or data structures by keys
- but [complex operations](#) like ZREVRANGEBYSCORE.
- INCR & co (good for rate limiting or statistics)
- Bit operations (for example to implement bloom filters)
- Has sets (also union/diff/inter)
- Has lists (also a queue; blocking pop)
- Has hashes (objects of multiple fields)
- Sorted sets (high score table, good for range queries)
- Lua scripting capabilities (!)
- Has transactions (!)
- Values can be set to expire (as in a cache)
- Pub/Sub lets one implement messaging

Best used: For rapidly changing data with a foreseeable database size (should fit mostly in memory).

For example: Stock prices. Analytics. Real-time data collection. Real-time communication. And wherever you used memcached before.

Hbase, Cassandra

- Hbase :
 - HBase는 아파치 하둡(Hadoop) 프로젝트의 하나로 개발
 - 구글의 빅테이블과 같은 기능을 제공하는 HDFS(Hadoop Distributed Filesystem)상에서 동작
- 카산드라
 - 칼럼(열) 기반 데이터베이스라 불리는 구글의 빅테이블 데이터 모델 기반
 - 카산드라의 데이터 모델은 여러 개의 칼럼(열)과 행, 칼럼 패밀리 및 키스페이스(KeySpace, 키가 가질 수 있는 값의 집합)로 구성



Clones of Google's Bigtable

HBase (V0.92.0)

- **Written in:** Java
- **Main point:** Billions of rows X millions of columns
- **License:** Apache
- **Protocol:** HTTP/REST (also Thrift)
- Modeled after Google's BigTable
- Uses Hadoop's HDFS as storage
- Map/reduce with Hadoop
- Query predicate push down via server side scan and get filters
- Optimizations for real time queries
- A high performance Thrift gateway
- HTTP supports XML, Protobuf, and binary
- JRuby-based (JIRB) shell
- Rolling restart for configuration changes and minor upgrades
- Random access performance is like MySQL
- A cluster consists of several different types of nodes

Best used: Hadoop is probably still the best way to run Map/Reduce jobs on huge datasets. Best if you use the Hadoop/HDFS stack already.

For example: Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement.

Cassandra (1.2)

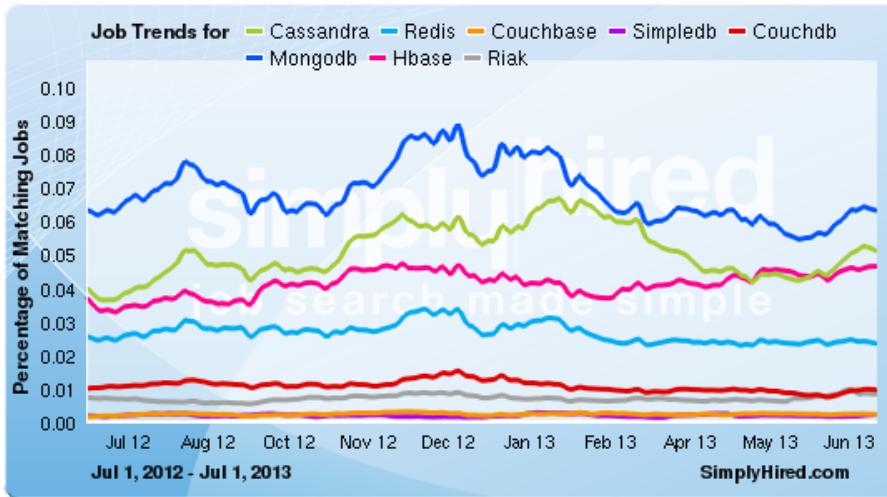
- **Written in:** Java
- **Main point:** Best of BigTable and Dynamo
- **License:** Apache
- **Protocol:** Thrift & custom binary CQL3
- Tunable trade-offs for distribution and replication (N, R, W)
- Querying by column, range of keys (Requires indices on anything that you want to search on)
- BigTable-like features: columns, column families
- Can be used as a distributed hash-table, with an "SQL-like" language, CQL (but no JOIN!)
- Data can have expiration (set on INSERT)
- Writes can be much faster than reads (when reads are disk-bound)
- Map/reduce possible with Apache Hadoop
- All nodes are similar, as opposed to Hadoop/HBase
- Very good and reliable cross-datacenter replication

Best used: When you write more than you read (logging). If every component of the system must be in Java. ("No one gets fired for choosing Apache's stuff.")

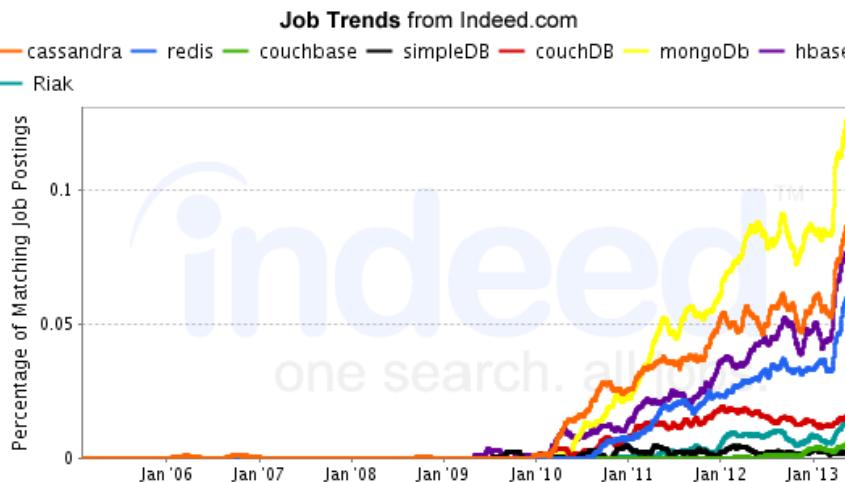
For example: Banking, financial industry (though not necessarily for financial transactions, but these industries are much bigger than that.) Writes are faster than reads, so one natural niche is data analysis.

※ Reference : <http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis>

Hot한 NoSQL 솔루션은?



- MongoDB가 증가 추세
- 2위는 cassandra,
- 3위는 hbase



어떤 NoSQL 솔루션을 사용해야 할까?

NOW COMPARING	Google Big Table	Apache Software Foundation CouchDB	Apache Software Foundation HBase	10gen MongoDB	Amazon SimpleDB
RELATED NOSQL DATABASES:	<ul style="list-style-type: none"> + Apache Apache Jackrabbit + Apache Software Foundation Cassandra + Hypertable Inc Hypertable 				 

KEY DETAILS					
Database Name	Big Table	CouchDB	HBase	MongoDB	SimpleDB
Developer	Google	Apache Software Foundation	Apache Software Foundation	10gen	Amazon
License Type	<ul style="list-style-type: none"> ✗ AGPL ✗ Apache ✗ Open Source ✓ Proprietary 	<ul style="list-style-type: none"> ✗ AGPL ✓ Apache ✓ Open Source ✗ Proprietary 	<ul style="list-style-type: none"> ✗ AGPL ✓ Apache ✓ Open Source ✗ Proprietary 	<ul style="list-style-type: none"> ✓ AGPL ✗ Apache ✓ Open Source ✗ Proprietary 	<ul style="list-style-type: none"> ✗ AGPL ✗ Apache ✗ Open Source ✓ Proprietary
Price	\$0				
Initial Release	2005	2005	2010	2009	2007
Project Url	http://labs.google.com/paper	http://couchdb.apache.org/	http://hbase.apache.org/	http://www.mongodb.org/	http://aws.amazon.com/simpledb

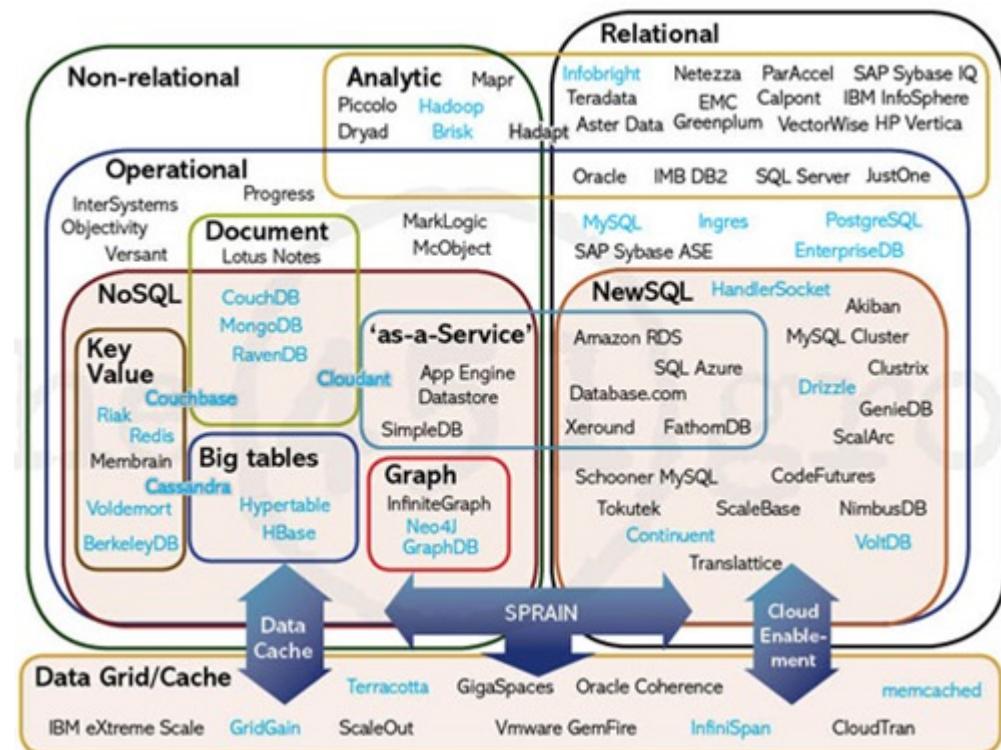
※ Reference : http://farahsuga.blogspot.kr/2012_03_01_archive.html

NoSQL vs. RDBMS

- 둘간의 차이점
- 왜 NoSQL을 사용하는가?

NoSQL (Not only SQL)

- NoSQL
 - High Expandability
 - Easy to Use in conventional load balanced clusters
 - Persistent data
 - Scale to available memory
 - No fixed schemas
 - individual query systems
 - ACID (node of cluster)



※ Reference : <http://cdi-mdm.blogspot.kr/2011/07/nosql-newsq...>

NoSQL/RDBMS 사용은 언제?

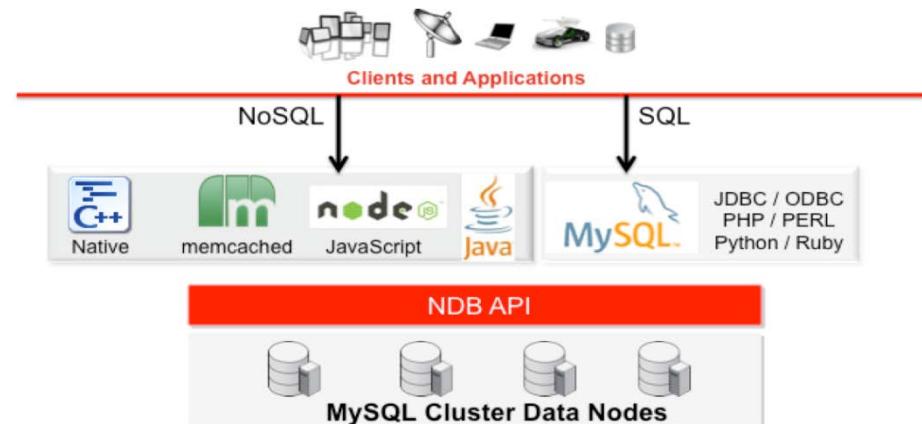
- 매우 큰 데이터 볼륨
 - RDBMS가 감당할 정도의 cost를 맞추어 가면서 사용하는 큰 traffic의 scale을 지원 가능한가?
 - 메인 데이터 저장소에 저장되지는 않지만, 임시적으로 사용하는 데이터를 많이 다루어야 하는가?
- 텍스트를 비롯한 많은 이미지들과 같은 큰 오브젝트들을 많이 다루어야 하는가?
(Google, Amazon, Yahoo, Facebook ……)
- 매우 큰 데이터 스케일링을 지원하기 위한 스키마 유연성이 필요한가?

Web App.은 어떤 걸 필요로 하나?

- 다양한 요구
 - 빠른 응답시간과 예측가능한 시간
 - 확장성, 물론 저렴하게….
 - 높은 가용성
 - 유연하게 변경가능 스키마, 다양하게 구성된 데이터
 - 여러 군데, 다양한 지역의 데이터 센터 지원
- 다음이 필요할까?
 - 트랜잭션, 일관성, 무결성, 복잡 쿼리

SQL과 NoSQL

- MySQL Cluster
 - RDBMS와 NoSQL을 혼용해서 사용
 - 비용 감소, 위험 감소, 복잡도 지원
 - 오토 샤딩, 스케일-아웃 지원
 - 온라인 스케일링과 스키마 변경 지원
 - ACID 호환, 복잡 쿼리 지원



※ Reference :

<http://www.mysql.com/products/cluster/nosql.html>

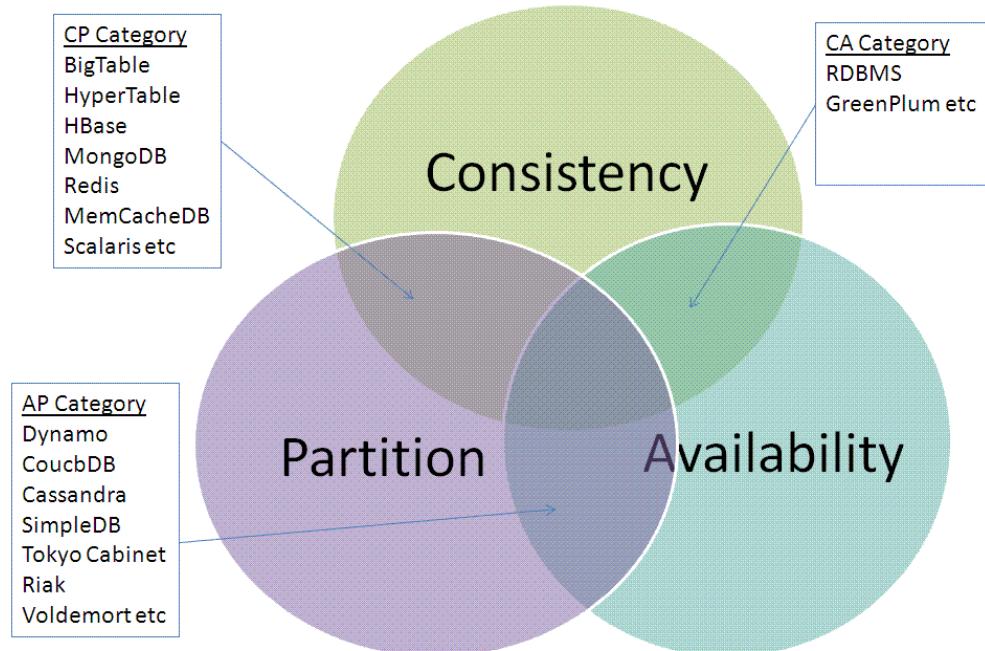
NoSQL, Distribution systems (1/2)

- 무엇이 필요한가?
 - C
 - Consistency : 수행 이후에 일관적인 상태를 가질 것.
 - 각 클라이언트들은 동일한 값을 봐야 함
 - A
 - Availability : 시스템은 항상 on, downtime은 No.
 - 모든 클라이언트들은 유효한 값을 가질 수 있어야.
 - 계속해서 읽고, 쓰고.
 - P
 - Partition tolerance : subset으로 나누어지더라도 계속 동작
 - 물리적인 네트워크 파티션을 거쳐서 잘 동작해야.

NoSQL, Distribution systems (2/2)

- CAP Theorem (E.Brewer, N. Lynch)
 - 3가지 중 두 가지만 만족해도 유효.

- CA
- CP
- AP



※ Reference : <http://amitpiplani.blogspot.kr/2010/05/u-pick-2-selection-for-nosql-providers.html>

MongoDB로의 전환

- RDBMS → NoSQL로의 전환 비용
 - 코드 재작성 비용
 - App. 구조 재정의 비용
 - 테스트 재수행 비용
 - 개발자와 운영자 재교육 비용
 - 관련 도구 재개발 및 프로세스 재정립 비용

SQL문과 MongoDB 간의 차이점 (1/3)

- Terms

SQL 사용 용어	MongoDB 사용 용어
데이터베이스(database)	데이터베이스(database)
테이블(table)	컬렉션(collection)
행(row)	문서(document) 또는 BSON 문서
열(column)	필드(field)
색인(index)	색인(index)
테이블 조인(table joins)	임베디드 문서 & 링킹(linking)
기본(주) 키(primary key, 유일한 고유 칼럼)	기본(주) 키(primary key, _id 필드 자동 생성)
집합(aggregation, 예: group by)	집합(aggregation) 프레임워크

SQL문과 MongoDB 간의 차이점 (2/3)

- Insert

SQL 문	MongoDB 문
<pre>INSERT INTO users(user_id, age, status) VALUES ("bcd001", 45, "A")</pre>	<pre>db.users.insert({ user_id: "bcd001", age: 45, status: "A" })</pre>

SQL문과 MongoDB 간의 차이점 (3/3)

- Select

SQL 문	MongoDB 문
SELECT * FROM users	db.users.find()
SELECT * FROM users WHERE age > 25	db.users.find({ age: { \$gt: 25 } })
SELECT * FROM users WHERE age < 25	db.users.find({ age: { \$lt: 25 } })
SELECT * FROM users WHERE age > 25 AND age <= 50	db.users.find({ age: { \$gt: 25, \$lte: 50 } })

1

NoSQL 개념

NoSQL ?

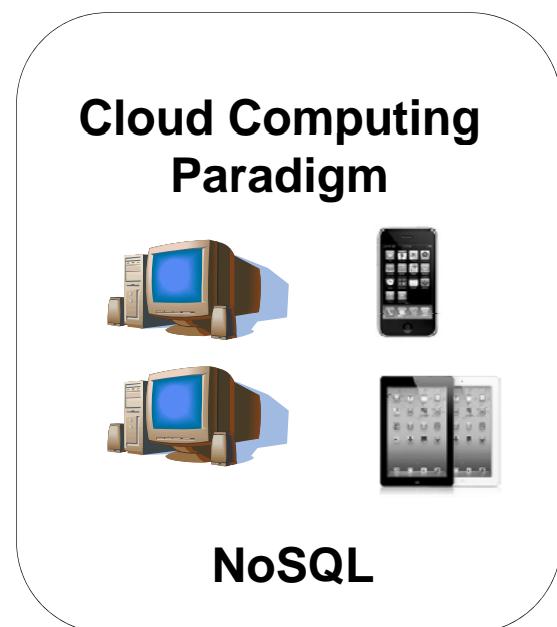
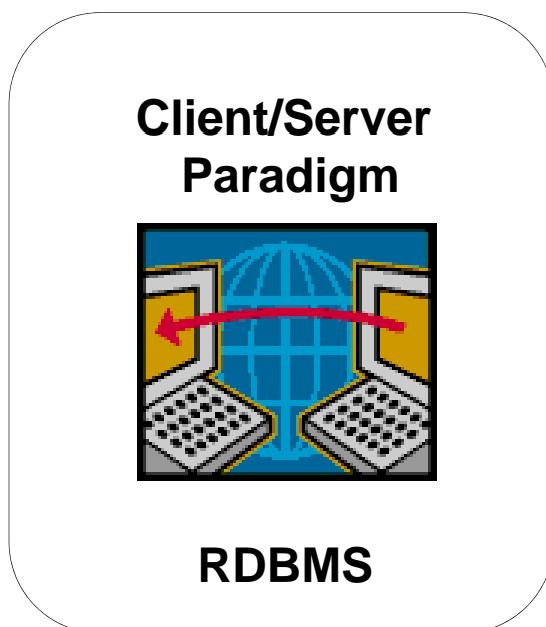
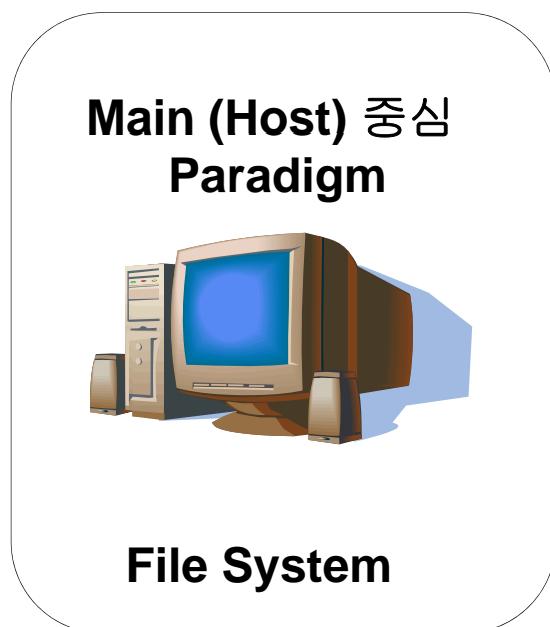
No SQL

Not Only SQL



**Non-Relational
Operational Database
SQL**

NoSQL의 시대적 요구



NoSQL의 장점

1. 클라우드 컴퓨팅 환경에 적합하다.

- 1) **Open Source**이다.
- 2) 하드웨어 확장에 유연한 대처가 가능하다.
- 3) 무엇보다 **RDBMS**에 비해 저렴한 비용으로 분산 처리와 병렬 처리가 가능하다.

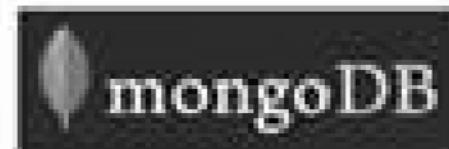
2. 유연한 데이터 모델이다

- 1) 비정형 데이터 구조 설계로 설계 비용 감소
- 2) 관계형 데이터베이스의 **Relationship**과 **Join** 구조를 **Linking**과 **Embedded**로 구현하여 성능이 빠르다.

3. Big Data 처리에 효과적이다

- 1) **Memory Mapping** 기능을 통해 **Read/Write**가 빠르다.
- 2) 전형적인 **OS**와 **Hardware**에 구축할 수 있다.
- 3) 기존 **RDB**와 동일하게 데이터 처리가 가능하다.

DBMS for NoSQL



NoSQL 제품군

1. Key-Value Database

- 1) Amazon's Dynamo Paper
- 2) Data Model : Collection of K-V pairs
- 3) 제품유형 : Riak, Voldemort, Tokyo*

3. Document Database

- 1) Lotus Notes
- 2) Data Model : Collection of K-V collection
- 3) 제품유형 : Mongo DB, Cough DB

2. BigTable Database

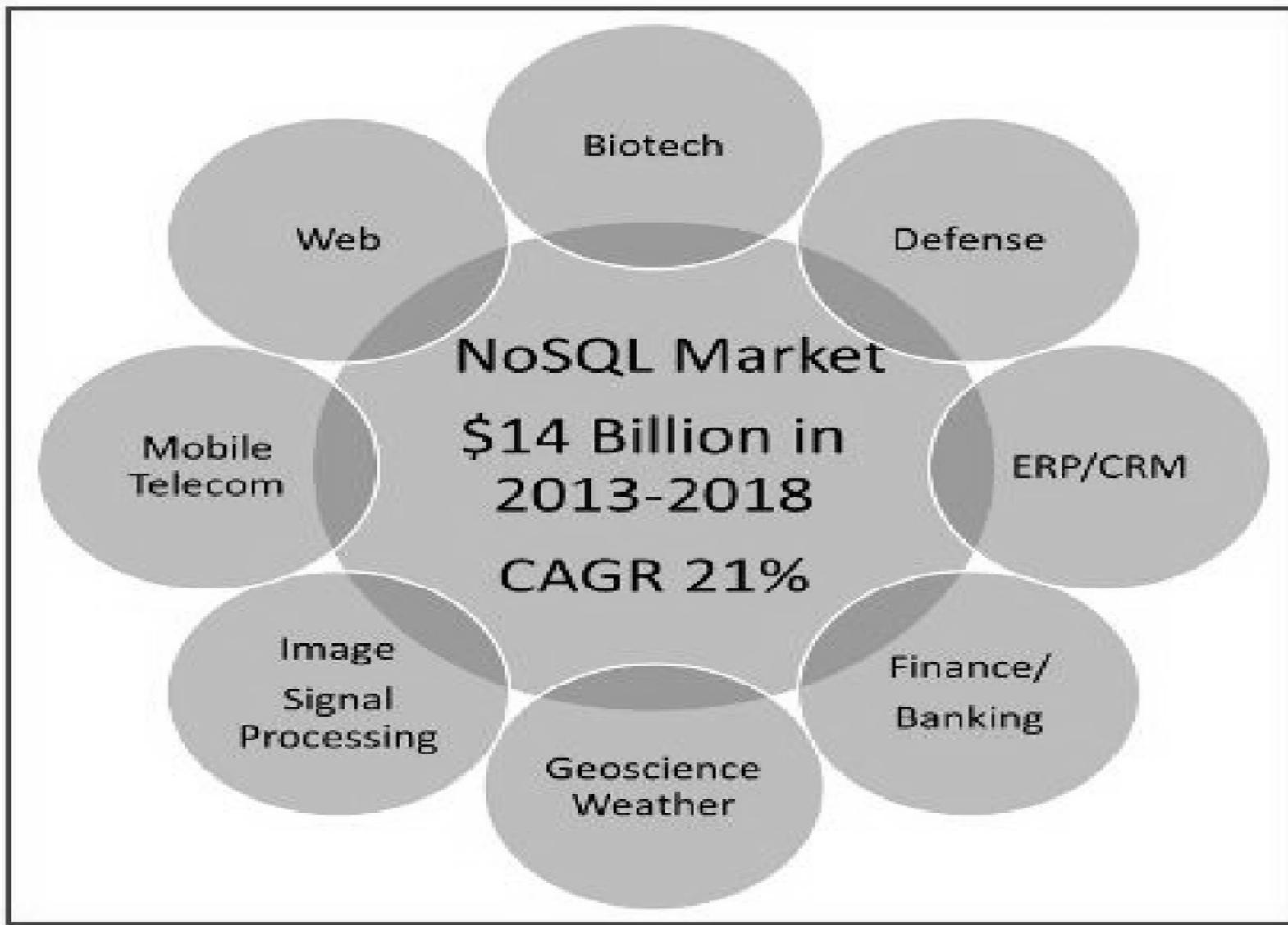
- 1) Google's BigTable paper
- 2) Data Model : Column Families
- 3) 제품유형 : Hbase, Casandra, Hypertable

4. Graph Database

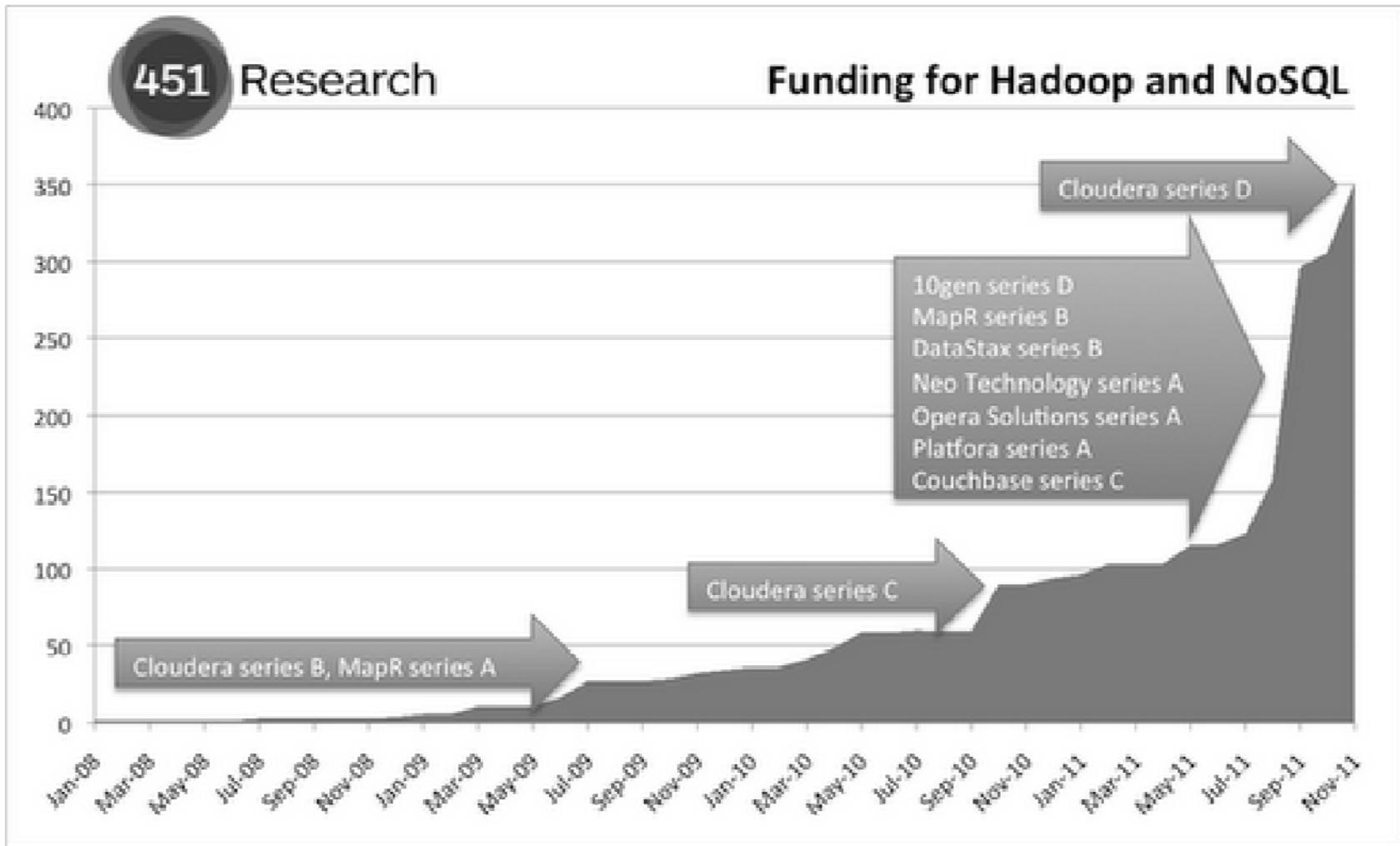
- 1) Euler & Graph Theory
- 2) Data Model : nodes, rels, K-V on both
- 3) 제품유형 : AllegroGraph, Sones

* Availability(유용성), Consistency(일관성), Partitioning(파티션ning)에 따른 제품군 구분

NoSQL 시장 동향

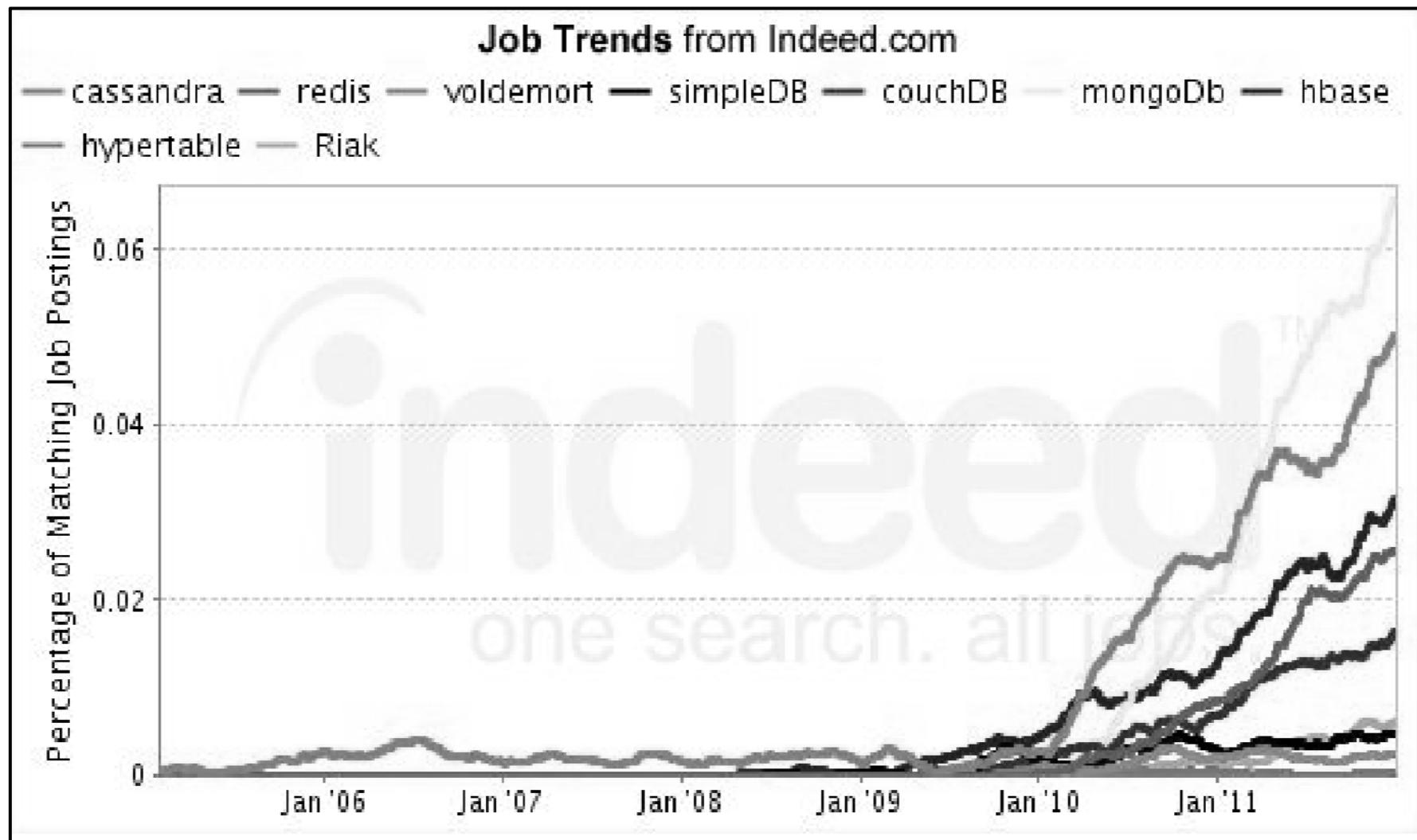


NoSQL Funding 동향



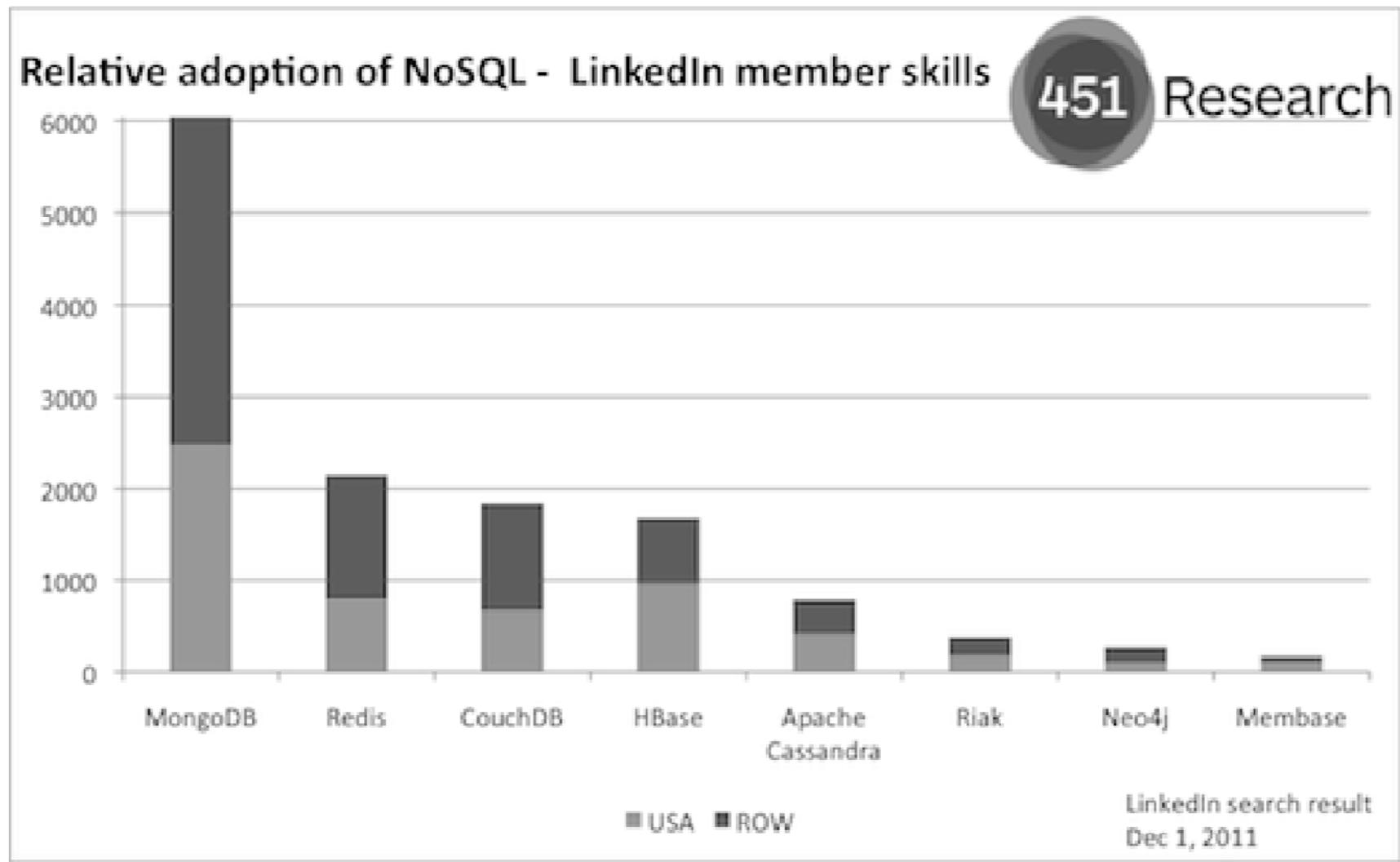
참조자료 : 2011년 7월 가드너 그룹

NoSQL 관련 직무 동향



참조자료 : indeed.com

NoSQL 관련 LinkedIn 멤버 동향



MongoDB Job 동향



* 2012년 6월 indeed.com 통계

▶ [Post on your blog](#)

Top Job Trends

- 1. [HTML5](#)
- 2. [MongoDB](#)
- 3. [iOS](#)
- 4. [Android](#)
- 5. [Mobile app](#)
- 6. [Puppet](#)
- 7. [Hadoop](#)
- 8. [jQuery](#)
- 9. [PaaS](#)
- 10. [Social Media](#)

NoSQL 제품별 평가 결과

평가 기준	Tokyo*Cabinet * Tokyo Tyrant	Berkely DB Memcache DB	Voldemort BDB JE	REDIS	MongoDB
Write (Small Data Set)	★★★★★	★★	★★	★★★★★	★★★★★
Write (Large Data Set)	★	★	★	★	★★★★
Random Read (Small Data Set)	★★★★★	★★	★★	★★★★★	★★★★★
Random Read (Large Data Set)	★★	★★	★	★★★★	★★★★
Speed 일관성	★	★★★★	★★	★★★★★	★★★★★
Storage 효율성	★★★★★	★★	★	★★	★★★★
Horizontal 확장성	★★★★★	★★★★	★★	★★★★	★★★★★
Manageability (관리성)	★★★★★	★★★★★	★★	★★	★★★★★
Stability (안정성)	★★★★★	★★★★★	★★	★★★★	★★★★★
Community Support	★★	★	★	★★	★★★★★

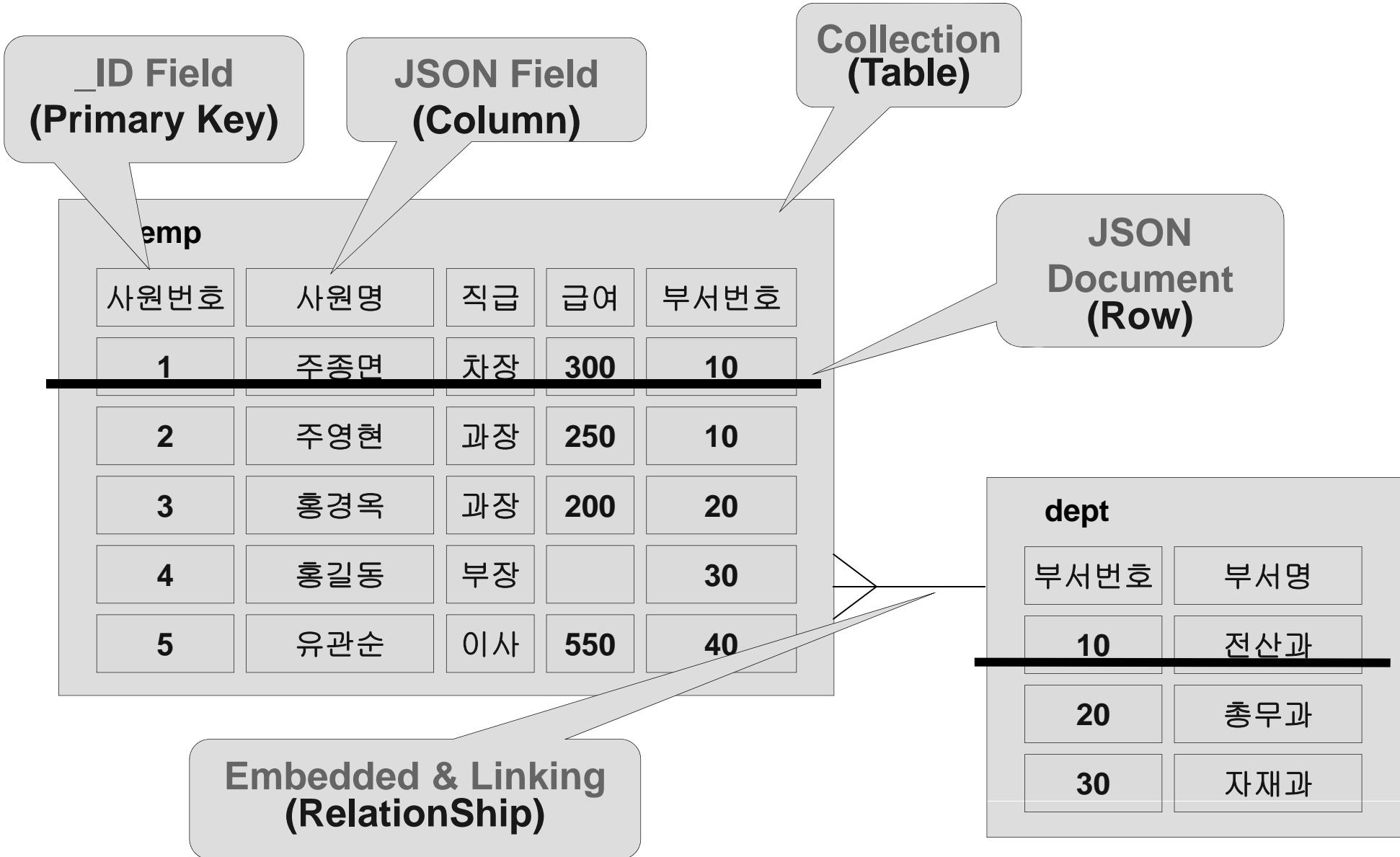
* 2011년 PerfectMarket 자료 참조

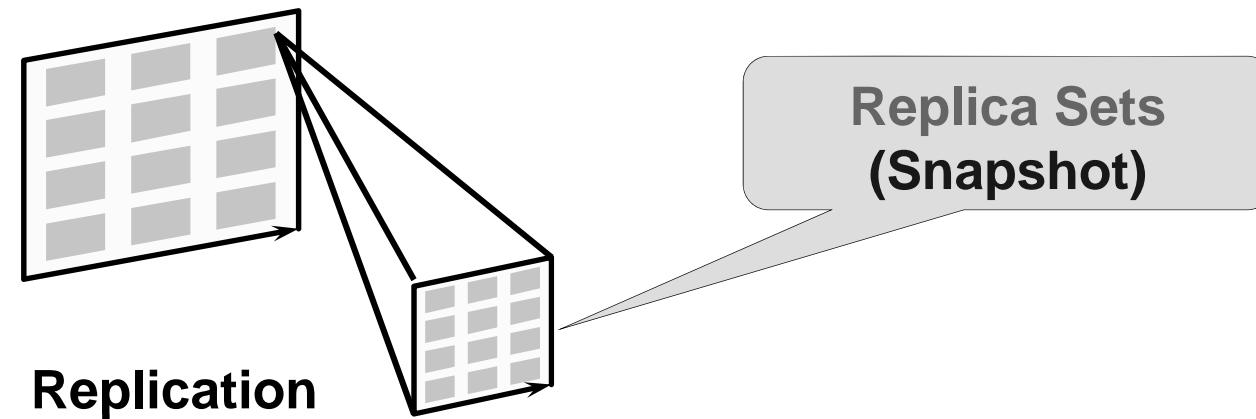
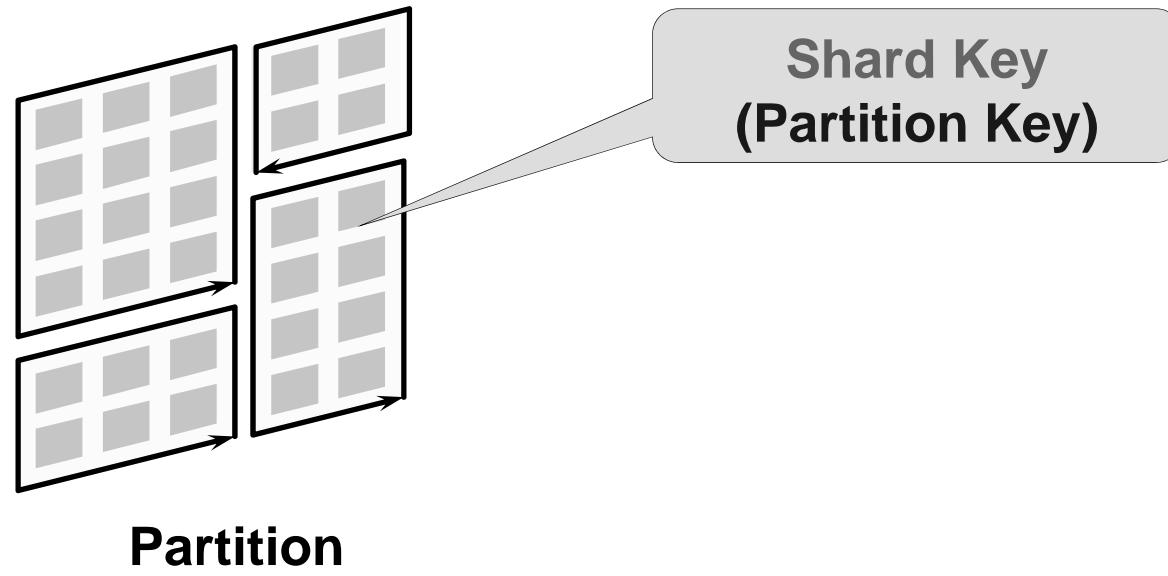
MongoDB 주요 특징

- 1) **Humongos**라는 회사의 제품 명이었으며 현재 **10gen**으로 회사명이 변경되었다.
- 2) **Document(JSON Type)**기반의 데이터 저장구조를 제공한다.
(경량의 데이터 교환 형식인 **JSON**(JavaScript Object Notation) 타입을 기반으로 하며 **JavaScript Programming Language , Standard ECMA-262 3rd Edition-1999** 근거로 함)
- 3) **Replica(복제)/Shard(파티셔닝)** 기능을 제공한다.
- 4) **MapReduce(분산/병렬처리)** 기능을 제공한다.
- 5) **CRUD(Create, Read, Update, Delete)** 위주의 다중 트랜잭션 처리도 가능하다.
- 6) **Memory Mapping** 기술을 기반으로 **Big Data** 처리에 탁월한 성능을 제공한다.

<http://www.10gen.com/what-is-mongodb>

Terminology



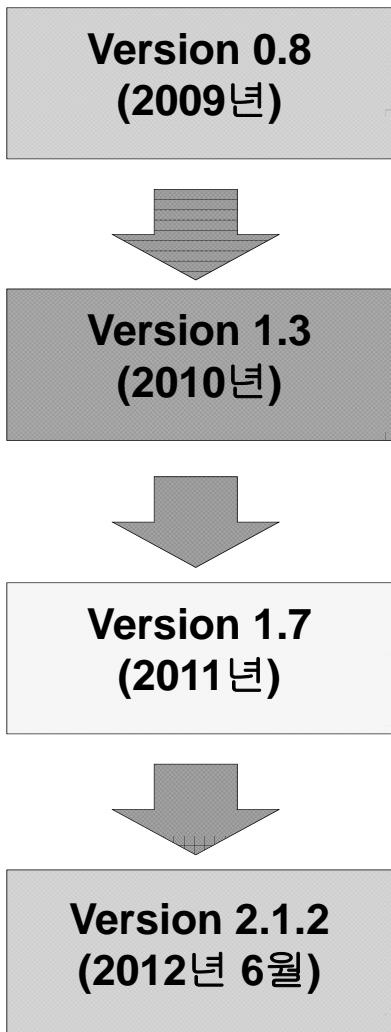


2

MongoDB 설치 및 데이터 처리

SQL & NoSQL

설치 환경 및 지원 드라이버



1) 설치 가능 플랫폼

- . Windows 32 bit / 64 bit
- . Linux 32 bit / 64 bit
- . Unix Solaris i86pc / 64 bit
- . Mac OS X-32bit / 64 bit

2) 지원 Language Driver

- . C / C# / C++
- . Java / Java Script
- . Perl / PHP / Python
- . Ruby / Erlang / Haskell / Scala

MongoDB의 시작

1) Mongod.exe

```
선택 관리자: 명령 프롬프트 - mongod --dbpath d:\mongodb\Wdb1
D:\mongodb\bin>
D:\mongodb\bin>mongod --dbpath d:\mongodb\Wdb1
Sun Jun 24 12:06:01 [initandlisten] MongoDB starting : pid=2572 port=27017
Sun Jun 24 12:06:01 [initandlisten] db version v2.0.6, pdflc version 4
Sun Jun 24 12:06:01 [initandlisten] git version: elc0cbc25003f0356aa4e3
Sun Jun 24 12:06:01 [initandlisten] build info: windows sys.netwindowswv
T_LIB_VERSION=1_42
Sun Jun 24 12:06:01 [initandlisten] options: { dbpath: "d:\mongodb\Wdb1" }
Sun Jun 24 12:06:01 [initandlisten] journal dir=d:/mongodb/db1/journal
Sun Jun 24 12:06:01 [initandlisten] recover begin
Sun Jun 24 12:06:01 [initandlisten] recover lsn: 12701841
Sun Jun 24 12:06:01 [initandlisten] recover d:/mongodb/db1/journal/j_0
Sun Jun 24 12:06:01 [initandlisten] recover skipping application of section seq:11/59/21 < lsn:12/61841
Sun Jun 24 12:06:01 [initandlisten] recover cleaning up
Sun Jun 24 12:06:01 [initandlisten] removeJournalFiles
Sun Jun 24 12:06:01 [initandlisten] recover done
Sun Jun 24 12:06:01 [initandlisten] preallocating a journal file d:/mongodb/db1/journal/prealloc.0
    10485760/1073741824  0%
    3145/280/10/3/41824  2%
    52428800/1073741824  4%
    /3400320/10/3/41824  6%
    83886080/1073741824  7%
    943/1840/10/3/41824  8%
    115343360/1073741824 10%
    130314880/1073741824 12%
    157288400/1073741824 14%
```

DBMS 종류	실행 파일	Client 툴
Mysql	Mysqld.exe	Mysql.exe
Oracle	Oracle.exe	Sqlplus.exe
MongoDB	Mongod.exe	Mongo.exe

2) Mongo.exe

```
관리자: 명령 프롬프트 - mongo
D:\mongodb\bin>
D:\mongodb\bin>mongo
MongoDB shell version: 2.0.6
connecting to: test
>
> help
    db.help()                      help on db methods
    db.mycoll.help()                help on collection methods
```

Collection 생성과 삭제

```
> db.createCollection ("emp", { chapped : false, size:8192 });
{ "ok" : 1 }                                     ← chapped : 해당 공간이 모두 사용되면 다시 처음부터
                                                재 사용할 수 있는 데이터 구조를 생성할 때
                                                size      : 해당 Collection의 최초 생성 크기 지정 가능

> show collections
emp
>
> db.emp.validate();                            ← Collection의 현재 상태 및 정보 분석
> {
    "ns" : "test.emp",
    "firstExtent" : "0:61000 ns:test.emp",
    "lastExtent" : "0:61000 ns:test.emp",
    "extentCount" : 1,
    "datasize" : 0,
    "nrecords" : 0,
    "lastExtentSize" : 8192,

> db.emp.renameCollection( "employees" )        ← 해당 Collection 이름 변경
> db.employees.drop();                         ← 해당 Collection 삭제
true
```

JSON 데이터 타입

- . 경량의 데이터 교환 형식인 **JSON**(**JavaScript Object Notation**) 타입을 근거로 하며 사람이 읽고 쓰기에 용이하며 기계가 분석하고 생성하기에 용이하다. (**Name**과 **Value**로 구성됨)
- . **JavaScript Programming Language**와 **Standard ECMA-262 3rd Edition-1999**을 근거로 한다. (*European Computer Manufacturers Association*)

{ **ename** : “주종면” }

Document Length

\x16\x00\x00\x00
\x06\x00\x00\x00

Value Length

Value Type

\x02ename\x00
주종면\x00\x00

JSON 타입과 데이터 저장

주문 (ORD) Document

```
> p = { ord_id : "2012-09-012345",
    customer_name : "Wonman & Sports",
    emp_name : "Magee",
    total : "601100",
    payment_type : "Credit",
    order_filled : "Y" }
```

```
> db.ord.save(p)
> db.ord.find()
```

* 하나의 **Document**는 반드시 유일한 값의 **Object_ID** 값이 부여된다.

데이터의 입력/수정/삭제

```
> db.emp.insert ({ eno : 1101, fname : "JIMMY" });

> db.emp.insert ({ eno : 1102, fname : "ADAM", lname : "KROLL" });

> db.emp.insert ({ eno : 1103, fname : "SMITH", job : "CLERK" });
```

```
> db.emp.update ({ eno:1101 }, { $set: { fname : "JOO" } } );

> db.emp.update ({ eno:1102 }, { $set: { job : "CHIEF" } } );

> db.emp.update ({ eno:1103 }, { $set: { lname : "STANFORD" } } );
```

```
> db.emp.remove ({ eno: 1101});

> db.emp.find().sort ({eno:-1});

{ "_id" : ObjectId("4fe6852f5642c534a77fbdb1"),
  "eno" : 1103, "fname" : "SMITH", "job" : "CERK", "lname" : "STANFORD" }

{ "_id" : ObjectId("4fe685195642c534a77fbdb0"),
  "eno" : 1102, "fname" : "ADAM", "job" : "CHIEF", "lname" : "KROLL" }
```

SQL & NoSQL 비교

SQL Statement

```
CREATE TABLE emp  
(empno Number, ename Number)
```

```
INSERT INTO emp VALUES(3,5)
```

```
SELECT * FROM emp
```

```
SELECT empno, ename FROM emp
```

```
SELECT * FROM emp WHERE empno=3
```

```
SELECT empno, ename  
FROM emp WHERE empno=3
```

```
SELECT * FROM emp  
WHERE empno=3 ORDER BY ename
```

NoSQL Statement

```
db.createCollection("emp")
```

```
db.emp.insert({empno:3, ename:5})
```

```
db.emp.find()
```

```
db.emp.find({}, {empno:1, ename:1})
```

```
db.emp.find({empno:3})
```

```
db.emp.find({empno:3}, {empno:1, ename:1})
```

```
db.emp.find({empno:3}).sort({ename:1})
```

Oracle & NoSQL 비교

```

SELECT deptno, job,
       SUM(sal) AS msum,    ← 부서별 급여합계
       COUNT(*) AS recs,   ← 부서별 인원수
       AVG(sal) AS mavg,   ← 평균급여금액
       MIN(sal) AS mmin,   ← 최소급여액
       MAX(CASE           ← 최대급여액
              WHEN sal > 1000
              THEN sal END) AS mmax
FROM emp
WHERE (hiredate > '01-01-1981'
       AND hiredate < '31-12-1983')
       AND sal > 800
GROUP BY deptno, job
HAVING min(sal) > 0
ORDER BY recs DESC
  
```

- 대상 Table과 Collection
- 검색 조건
- 검색 Column 또는 Field
- Aggregate 또는 Procedure Logic
- Aggregate 또는 Procedure Logic
- Aggregate Filter 또는 Sorting

```

db.runCommand ({ mapreduce: "emp",
query: {
  hiredate : { $gt : '01-01-1981', $lt : '31-12-1983' },
  sal      : { $gt : 800 } },
map: function() {
  emit( { d1 : this.deptno, d2 : this.job },
        { msum: this.sal, recs: 1, mmin: this.sal,
          mmax: this.sal > 1000 } );
},
reduce: function(key, vals)
{
  var ret = { msum:0, recs:0, mmin:0, mmax:0 };
  for (var i=0 ; i < vals.length; i++)
  {
    ret.msum += vals[i].msum;
    ret.recs += vals[i].recs;
    if (vals[i].mmin < ret.mmin) ret.mmin=vals[i].mmin;
    if (vals[i].mmax > 1000)    ret.mmax=vals[i].mmax;
  }
  return ret;
},
finalize: function(key, val)
{
  val.mavg=val.msum/val.recs;
  return val;
},
out: "result1",
verbose: true
});
db.result1.find
({ "value.mmin": { $gt:0 }}).sort({ "value.recs": 1});
  
```

INDEX 생성과 관리

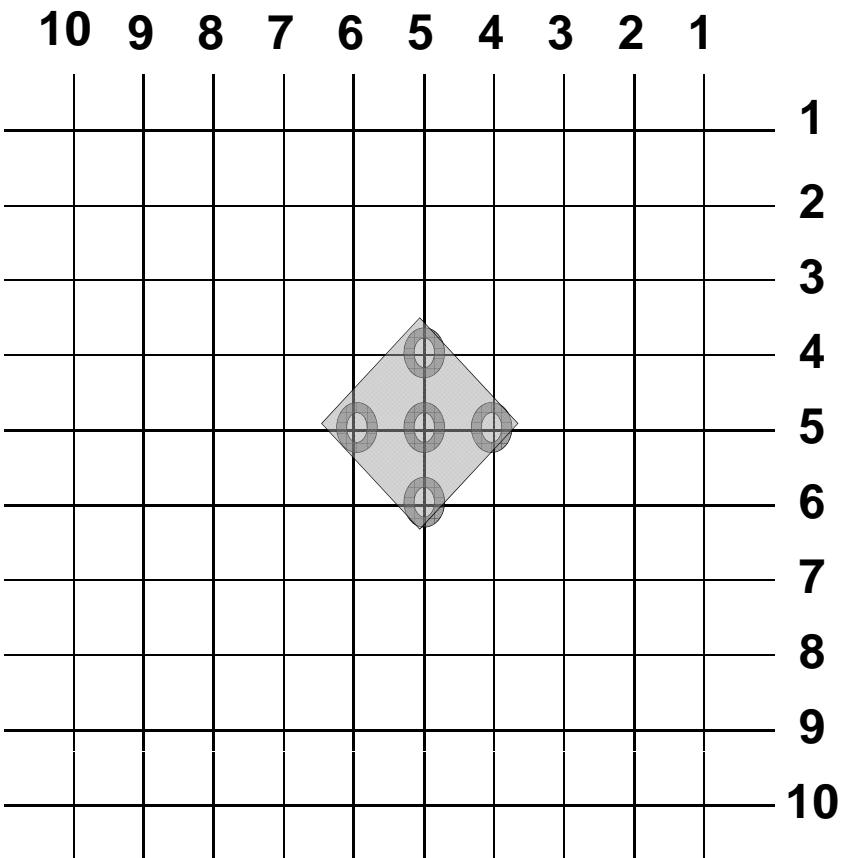
```
> db.emp.ensureIndex ({ eno: 1 }, { unique: true });
    ← 1 (Asc), -1 (Desc)
> db.emp.dropIndex ({ eno : 1 });
```

```
> db.emp.getIndexes()
{
    "v" : 1,
    "key" : {
        "eno" : 1
    },
    "unique" : true,
    "ns" : "test.emp",
    "name" : "eno_1"
}
```

```
> db.system.indexes.find()
{ "v" : 1, "key" : { "eno" : 1 },
  "unique" : true, "ns" : "test.emp", "name" : "eno_1" }
```

GeoSpatial INDEX

- . 좌표에 의해 구성되는 2차원 구조로 하나의 **Collection**에 하나의 **2D Index**를 생성할 수 있다.



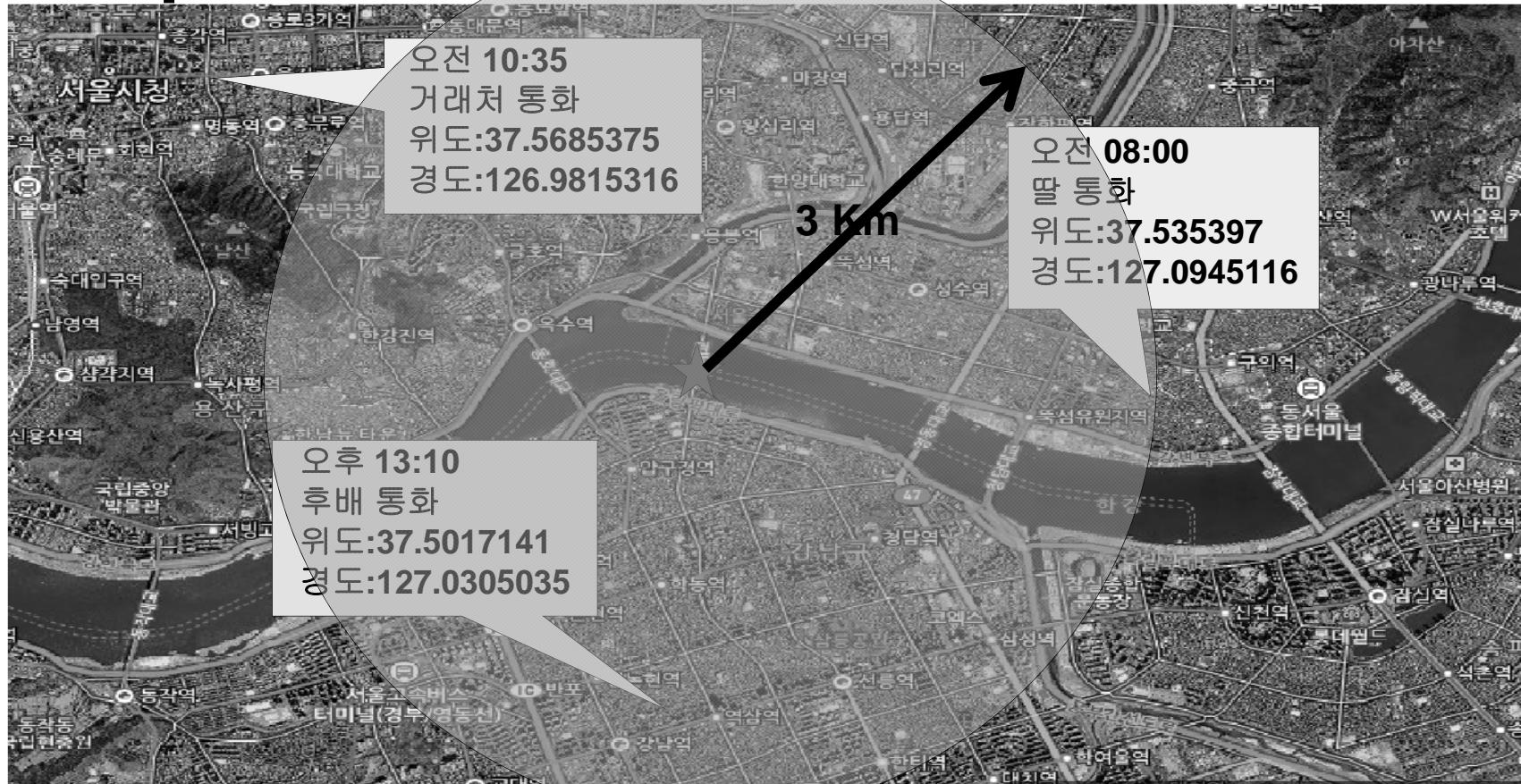
```
> for ( var i = 0; i < 100; i++)  
    db.square.insert ({  
        pos : [ i % 10, Math.floor( i / 10 ) ] } )  
  
> db.square.ensureIndex ({ pos : "2d" })  
  
> db.square.find({ pos : { $near : [5, 5] } }).limit(5)  
{ "pos" : [ 5, 5 ] }  
{ "pos" : [ 5, 4 ] }  
{ "pos" : [ 4, 5 ] }  
{ "pos" : [ 5, 6 ] }  
{ "pos" : [ 6, 5 ] }
```

Multi-Location Documents



```
> db.tel_pos.save ({ mobile_no : "01038641858",
    last_pos : [[ 127.0945116, 37.535397],
                [ 126.9815316, 37.5685375],
                [ 127.0305035, 37.5017141]]})
> db.tel_pos.ensureIndex( { last_pos : "2d" });
```

\$nearSphere



```
> db.tel_pos.find( { last_pos : { $nearSphere :  
[[ 127.0352915, 37.5360206], 3/6371 ] }}, { _id:0, last_pos : 0 } )
```

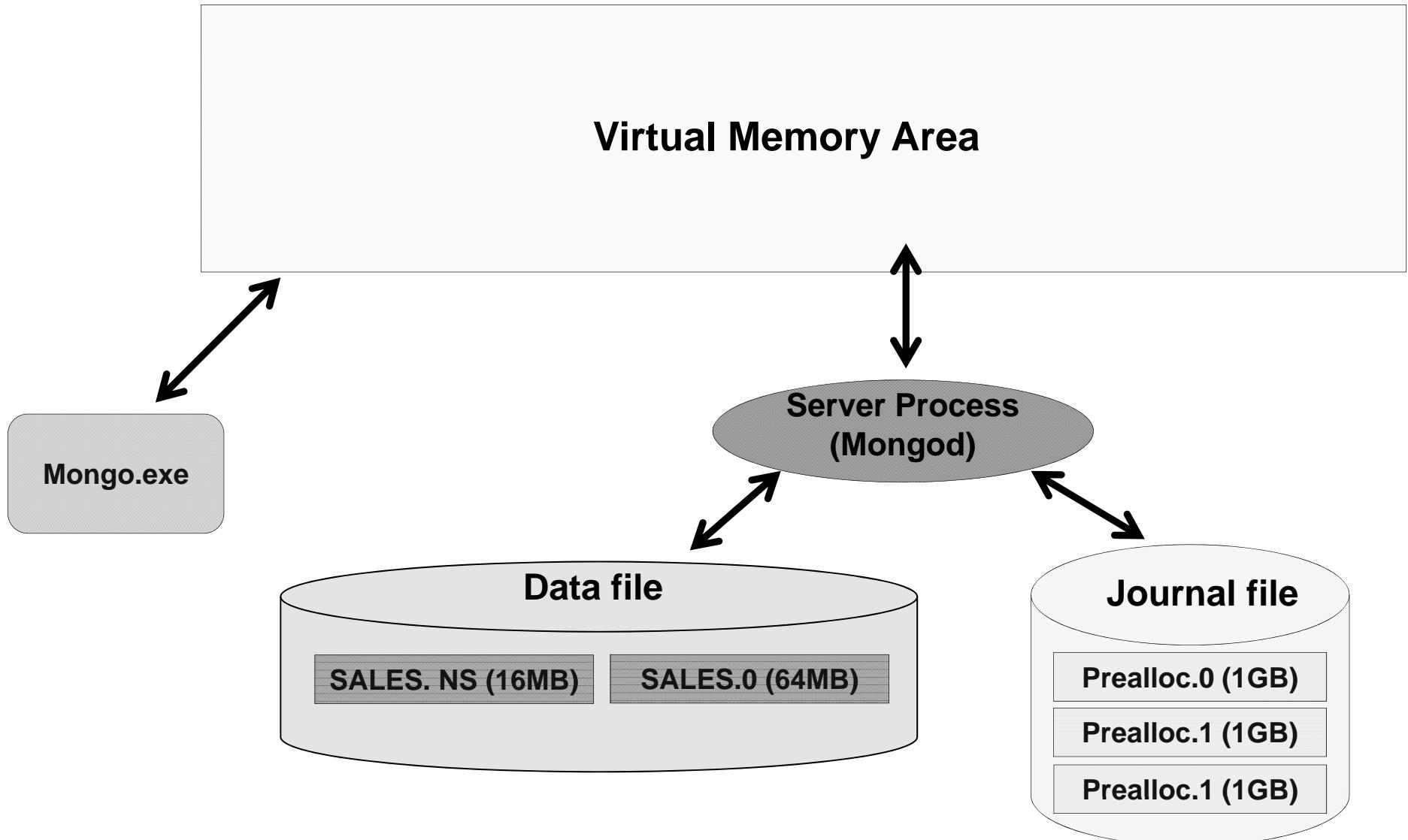
← 성수대교를 기준으로 반경 3 Km 이내

{ "mobile_no" : "01038641858" , last_pos : [127.0945116, 37.535397] } ← 딸과 통화 내역

{ "mobile_no" : "01038641858" , last_pos : [127.0305035, 37.5017141] } ← 후배와 통화 내역

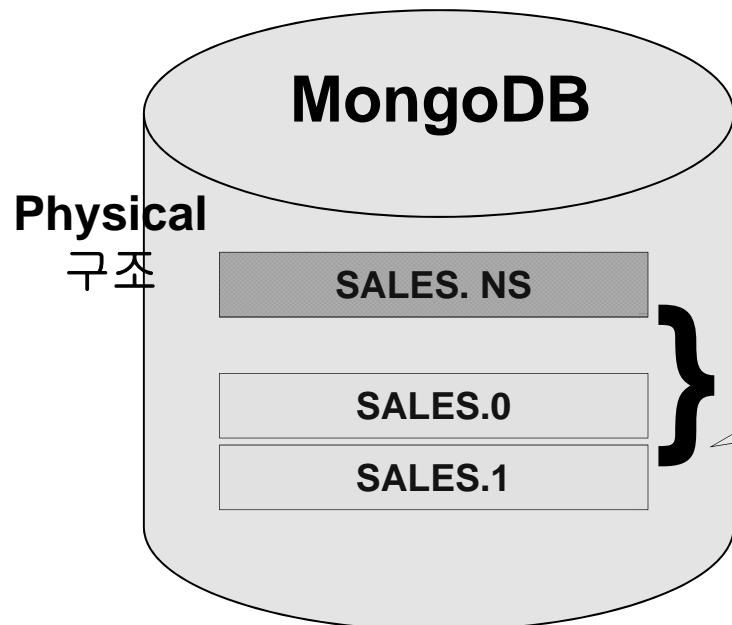
논리적 구조와 물리적 구조

MongoDB Architecture (Single Node)



Physical & Logical Architecture

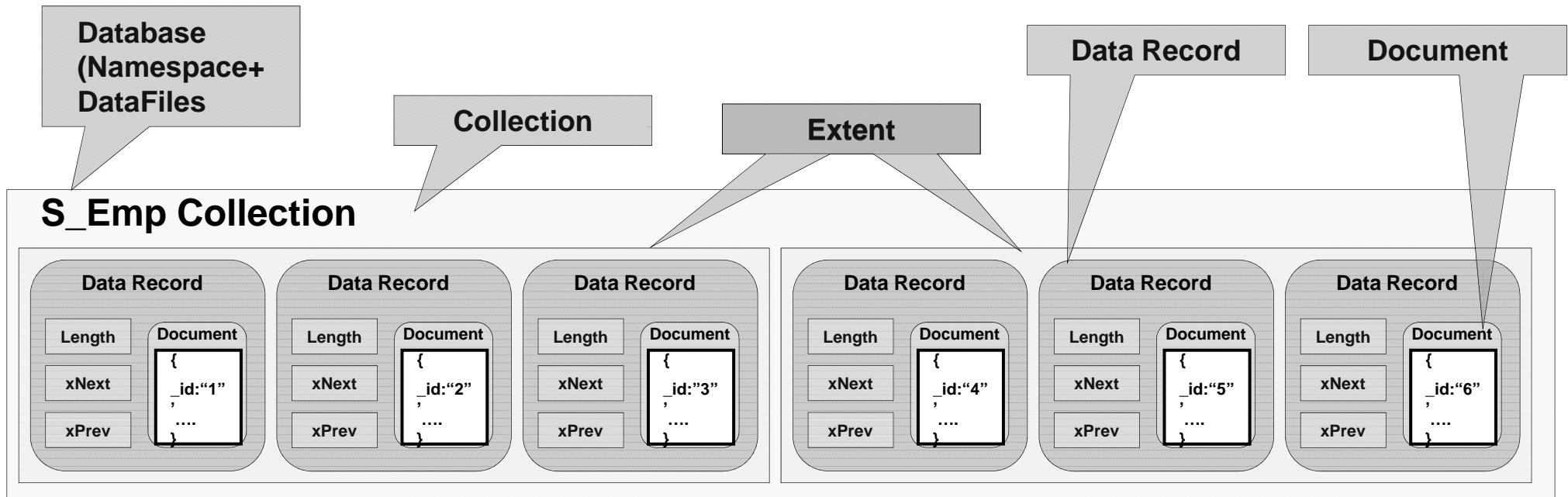
구조



```
Mongo
MongoDB shell version: 2.0.2
connecting to: test
>
> use SALES
switched to db SALES
>
>
> db.createCollection("s_emp",
{capped:false, size:100000});
```

- 1) 32 Bit에서 Namespace의 최초 크기는 16MB가 할당되며 DataFile은 32 MB 크기로 생성된다. Next 크기는 32MB의 배수로 증가하되 최대 크기는 2GB 이다. (64 Bit에서는 16MB, 64 MB 단위, Journing은 1GB)
- 2) Namespace에는 Collection의 First Extent와 LastExtent에 대한 정보와 Meta Data, 인덱스 , FreeList 정보가 저장된다.

Logical Architecture



- 1) MongoDB의 논리적 구조는 **Database → Collections → Extents → Data Records → Documents**로 구성된다.
- 2) **Collection** 크기는 최초 생성 시점에 결정되며 **Extent** 기본 크기는 8K이며 데이터 크기가 작은 경우에는 4K로 생성된다.(사용자에 의해 결정 가능)
(Ex) db.createCollection("s_emp", {capped:false, size:100000});

Structure Status

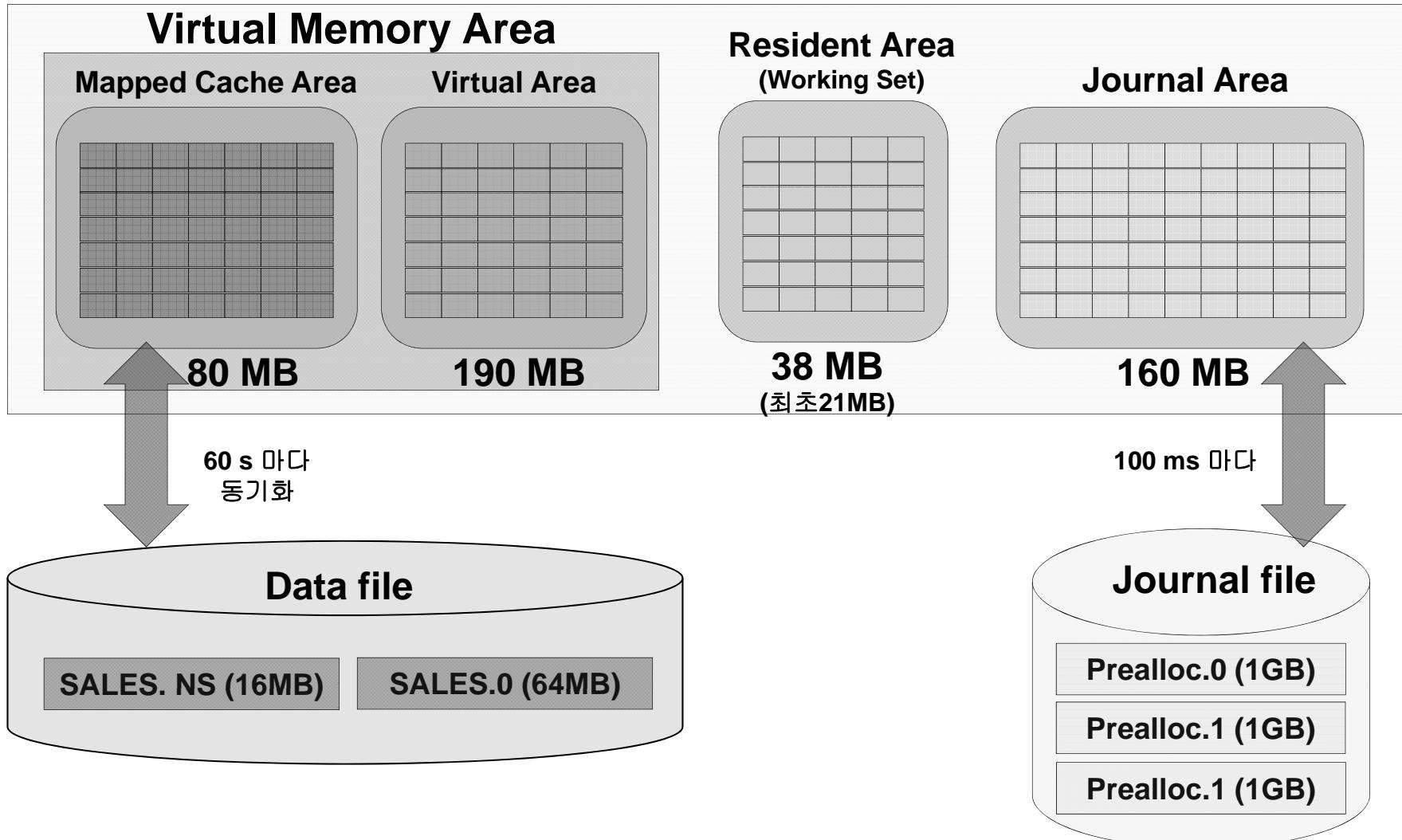
```
선택 관리자: 명령 프롬프트 - mongo_sales
> db.createCollection("s_emp");
[{"ok": 1}]
> db.s_emp.validate();
[
  {
    "ns": "sales.s_emp",
    "firstExtent": "0:30000 ns:sales.s_emp",
    "lastExtent": "0:30000 ns:sales.s_emp",
    "extentCount": 1,
    "datasize": 0,
    "nrecords": 0,
    "lastExtentSize": 8192,
    "padding": 1,
    "firstExtentDetails": {
      "loc": "0:30000",
      "xnext": "null",
      "xprev": "null",
      "nsdiag": "sales.s_emp",
      "size": 8192,
      "firstRecord": "null",
      "lastRecord": "null"
    },
    "deletedCount": 1,
    "deletedSize": 8016,
    "nIndexes": 1,
    "keysPerIndex": {
      "sales.s_emp._id_": 0
    },
    "valid": true,
    "errors": [],
    "warning": "Some checks omitted for speed. us",
    "ok": 1
  }
]
>
>
```

```
선택 관리자: 명령 프롬프트 - mongo_sales
> db.s_emp.insert({ _id: 1102, ename: "king" });
>
> db.s_emp.validate();
[
  {
    "ns": "sales.s_emp",
    "firstExtent": "0:30000 ns:sales.s_emp",
    "lastExtent": "0:30000 ns:sales.s_emp",
    "extentCount": 1,
    "datasize": 56,
    "nrecords": 1,
    "lastExtentSize": 8192,
    "padding": 1,
    "firstExtentDetails": {
      "loc": "0:30000",
      "xnext": "null",
      "xprev": "null",
      "nsdiag": "sales.s_emp",
      "size": 8192,
      "firstRecord": "0:300b0",
      "lastRecord": "0:300b0"
    },
    "deletedCount": 1,
    "deletedSize": 7944,
    "nIndexes": 1,
    "keysPerIndex": {
      "sales.s_emp._id": 1
    },
    "valid": true,
    "errors": [],
    "warning": "Some checks omitted for speed. us",
    "ok": 1
  }
]
```

메모리 구조

Memory Structure(64 Bit)

* 최초 약 470 MB



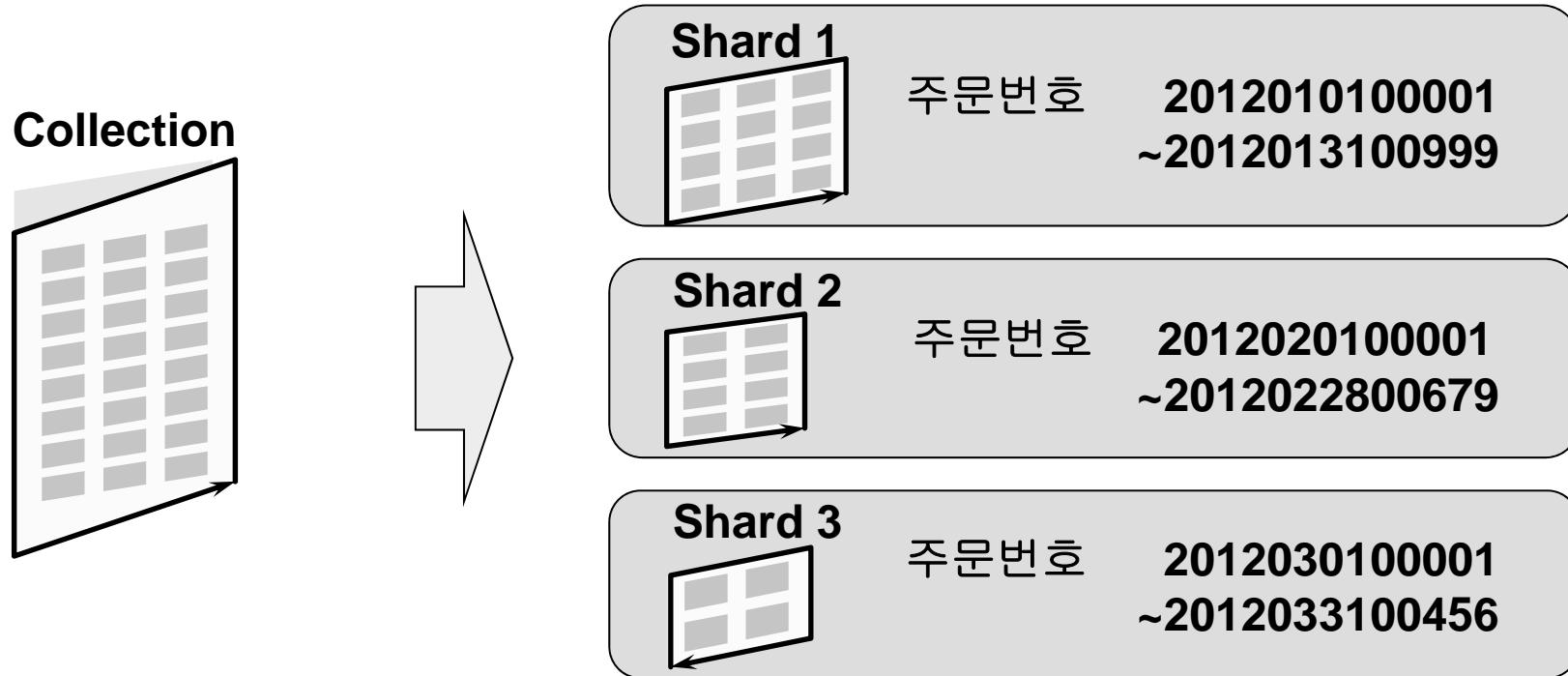
db.serverStatus().mem

적정 메모리 요구 사항

- 1) MongoDB는 **Direct Mapping** 기법에 의해 데이터를 처리하기 때문에 최초 메모리 크기는 생성되는 **Namespace**와 **Data-File**에 맞는 적정한 **Mapped Area** 크기가 요구된다.
(64 Bit의 경우 최초 Namespace 16MB + 첫 번째 Data-File 64MB)
- 2) 사용자의 데이터를 위한 **Virtual Memory**와 함께 MongoDB 서버를 원활하게 운영하기 위한 **Resident Area**와 **Journal Area**가 요구된다.
이 공간은 **Mapped Area** 크기에 따라 서버에 의해 동적으로 할당된다.
- 3) **Mapped Cache Area** 크기가 **2 GB**인 경우 요구되는 **RAM** 메모리는 약 **10 GB ~12 GB** 정도가 요구된다. **SYSTEM** 메모리가 부족한 경우 전체 공간의 **80%** 까지 자동으로 동적 할당되며 **Flushing**과 **Segmentation Fault**가 발생하여 성능 저하 현상이 발생할 수 있다. (**32 Bit**에서 최소 **4GB** 이상 요구)

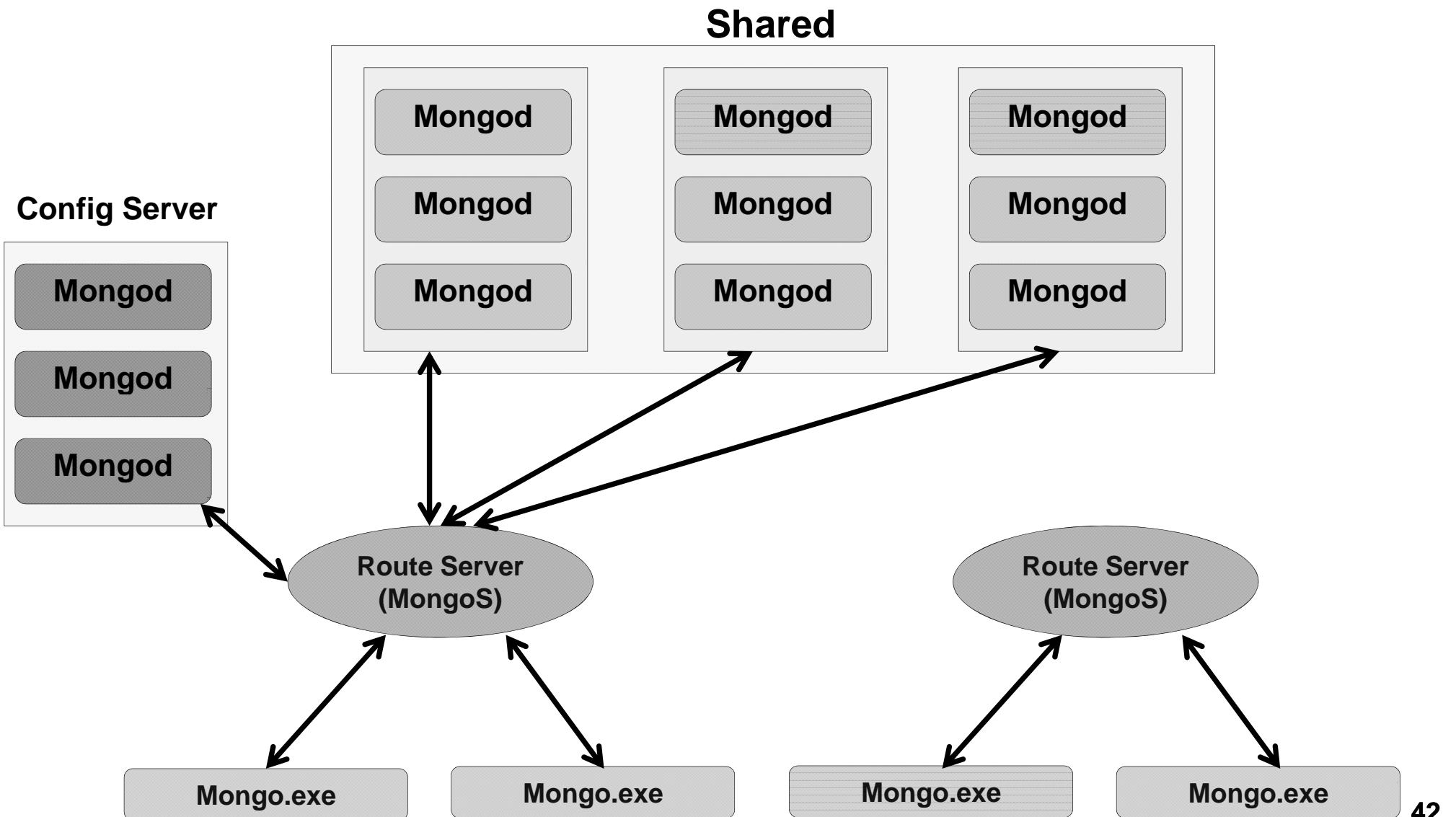
MongoDB 주요 기능

Sharding



- 1) Sharding의 가장 큰 목적은 파티셔닝을 통한 데이터 분산 처리와 성능 향상을 위한 **Load Balancing**이다.
- 2) 또한, 빅 데이터의 효율적 관리와 백업 및 복구 전략 수립을 위한 솔루션이기도 하다.

MongoDB Architecture (Multi Node)



Shard 환경 설정

1) MongoS 프로세스를 활성화 한다.

```
C:\> mongos --configdb 192.168.0.1:10000,192.168.0.2:10001,192.168.0.3:10002
```

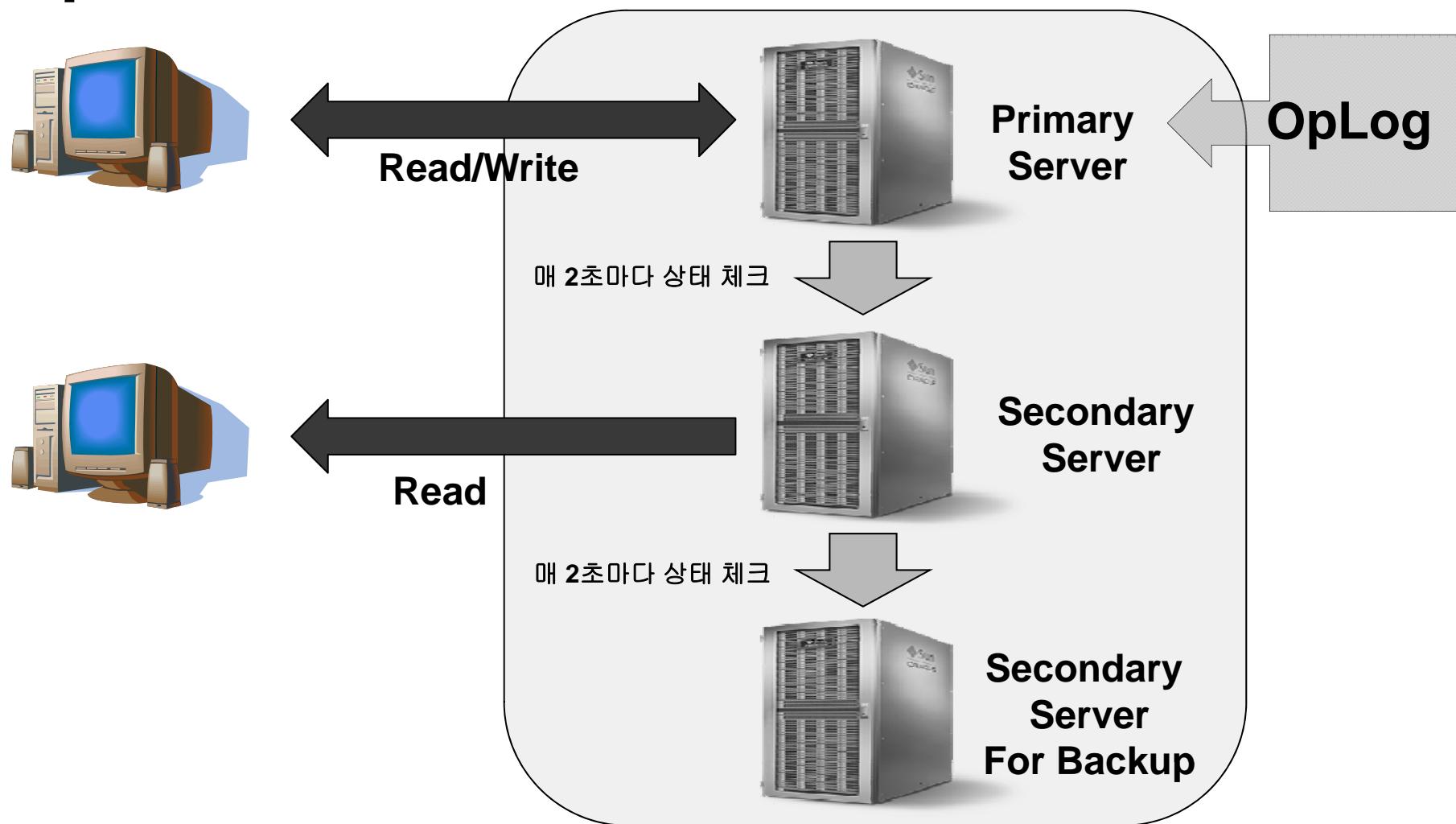
2) Config Server가 설치된 Node에서 각 Node가 상호 연결될 수 있도록 등록한다.

```
C:\> mongo localhost:27017/admin ← mongos에 접속하여 Node 정보 등록
mongos>
mongos> db.runCommand( { addshard : "192.168.0.1 : 10000" } ); ← Node1 등록
{ "shardAdded" : "shard0000", "ok" : 1 }
mongos>
mongos> db.runCommand( { addshard : "192.168.0.2 : 10001" } ); ← Node2 등록
{ "shardAdded" : "shard0001", "ok" : 1 }
mongos>
mongos> db.runCommand( { addshard : "192.168.0.3 : 10002" } ); ← Node3 등록
{ "shardAdded" : "shard0002", "ok" : 1 }
mongos>
mongos> db.runCommand( { enablesharding : "test" } ); ← 해당 DB Shard 기능 설정
{ "ok" : 1 }

> db.runCommand( { shardcollection : "test.things", key : { _id : 1 } } ); ← Shard Key 설정
{ "collectionsharded" : "test.things", "ok" : 1 }
```

Replica 기능

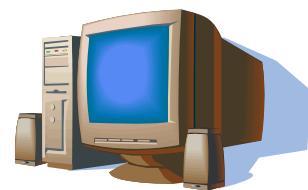
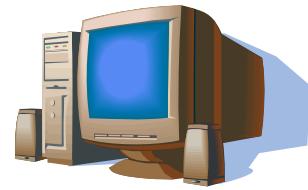
Replica Set



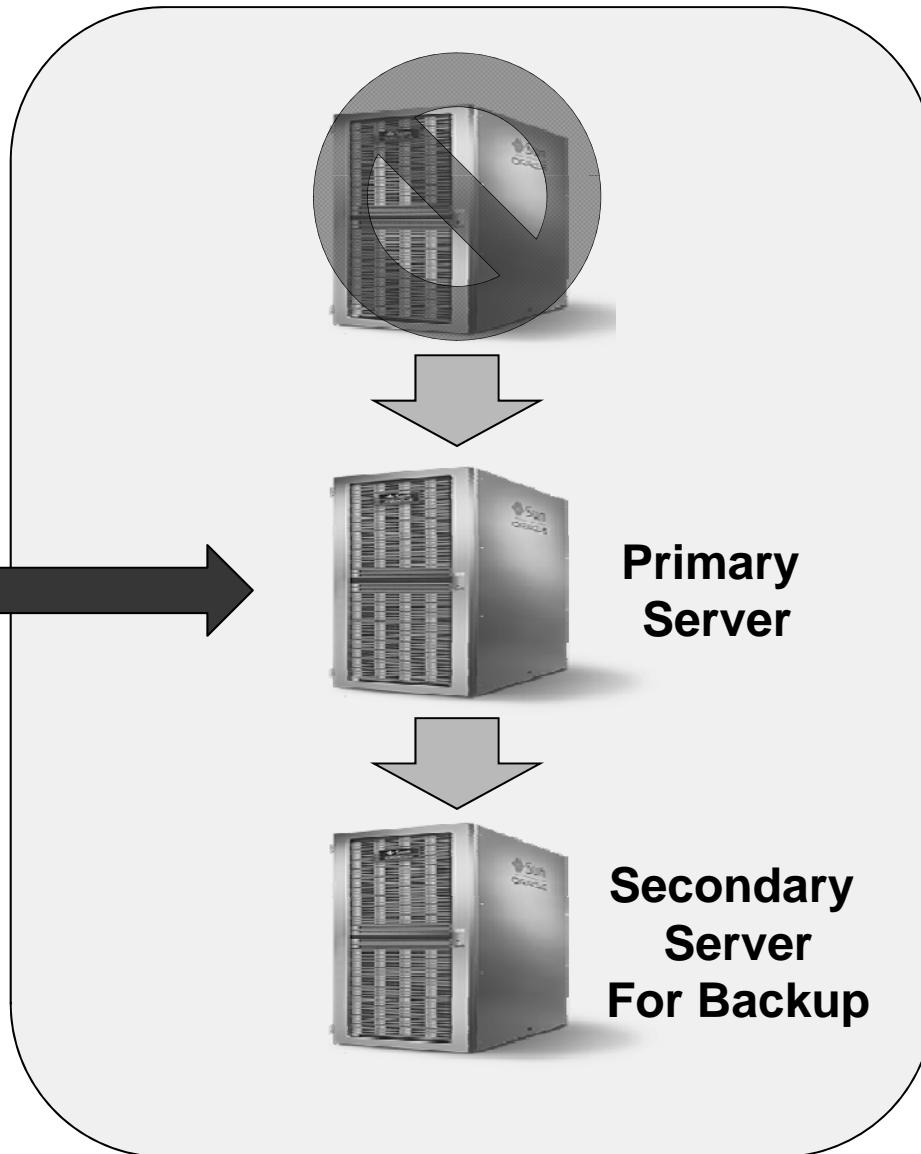
- 1) Heartbeat : 매 2초마다 **Secondary** 상태를 체크한다.
- 2) **Secondary**가 **Down** 되더라도 복제만 중지될 뿐 **Primary**에 대한 작업은 정상적이다.
- 3) **Primary**가 다운되면 **Secondary**가 **Primary**가 된다.
- 4) **OpLog**는 복제가 실패하는 경우를 위해 로그 정보를 저장해 준다.(기본 크기 1GB)

Fail Over-1

Replica Set

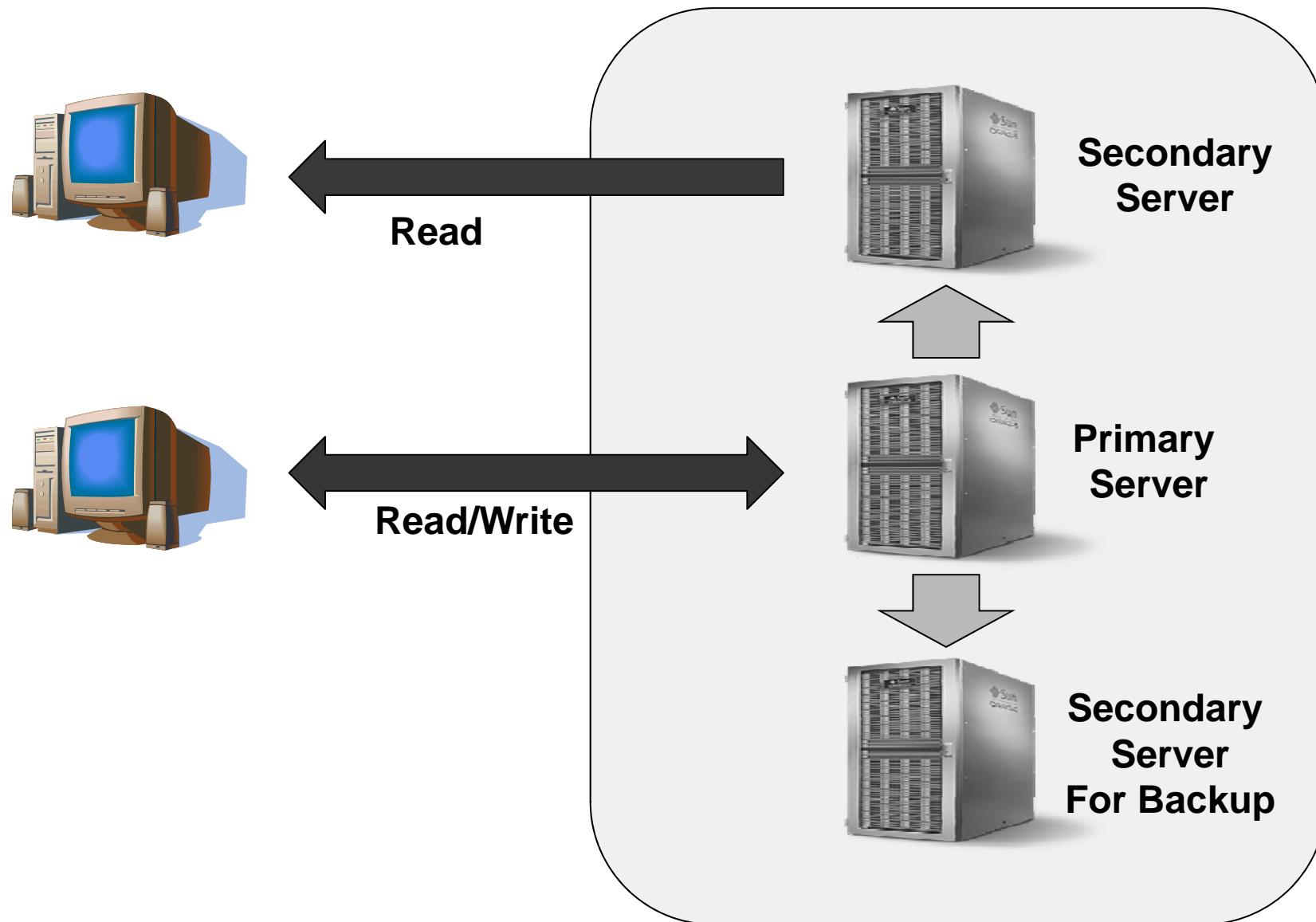


Read/Write



Fail Over-2

Replica Set



환경 설정

1) Primary DB에서 입력된 데이터가 Secondary DB에 복제될 수 있도록 데이터를 입력한다.

```
C:\> mongo 192.9.200.1:10000
> use test
> db.things.insert( { empno : 1101,ename : "james",dept : "account" } );
> db.things.find();

{"_id":ObjectId("4f03b6c6e5c6022a325f7181"),
 "empno":1101, "ename":"james", "dept":"account"}
```

2) Secondary DB에서 복제된 데이터를 확인한다.

```
C:\> mongo 192.9.200.2:10001
> use test
> db.things.find();      ← Primary DB에서 생성된 Collection과 Document

{"_id":ObjectId("4f03b6c6e5c6022a325f7181"),
 "empno":1101, "ename":"james", "dept":"account"}
```

보안/인증 방법

OS 인증 방식

- 1) 전형적인 인증 방식으로 **MongoDB**를 설치했던 **O/S** 계정의 권한으로 데이터베이스에 접속할 수 있다.
- 2) **MongoDB**에서는 해당 시스템의 **IP-Address**로 만 접속을 허용하는 **Network** 인증 방식을 제공한다.

(예) `mongod --bind_ip 192.168.0.10`

DB 인증 방식

- 1) **DBMS**의 가장 보편적인 인증 방식으로 미리 사용자 계정과 암호를 생성한 뒤 이 계정명과 암호를 정확히 입력한 사용자를 인증하는 방법이다.
(예) `db.addUser("jimmy", "joo")`
`db.auth("jimmy", "joo")`
- 2) **MongoDB**에 접속할 때 **DB** 인증 방식을 선택할 수 있다.
(예) `mongod --auth` 또는 `mongod --noauth`
- 3) **Replica Sets** 환경을 구현할 때 **Primary Server**와 **Secondary Server** 간에 인증을 위해 인증 **Key**가 요구된다.
(예) `mongod --keyFile`

3

MongoDB를 위한 Data Modeling

Data Modeling 핵심

- 1) **HOST** 환경의 파일 시스템은 프로세스 중심의 데이터 구조 설계였다면
클라이언트/서버 환경의 관계형 DB는 **Data** 중심의 설계를 지향하였다.
반면에 클라우드 컴퓨팅 환경의 **NoSQL**은 **Data**와 프로세스, 모두를
설계의 중심으로 둔다.
(Big Data의 수집 및 저장과 함께 유연성 있는 데이터 처리도 중요함)
- 2) **Rich Document Structure**
관계형 DB는 정규화를 통해 데이터 중복을 제거하며 무결성을 보장하는
설계 기법을 지향하지만 **NoSQL**은 데이터의 중복을 허용하며 비정규화된
설계를 지향한다. (**관계형 DB**가 요구되었던 당시와 달리 저장장치의
비약적 발전과 저렴한 가격 요인도 설계에 중요한 요소임)

- 3) 관계형 DB는 Entity 간의 Relationship을 중심으로 데이터의 무결성을 보장하지만 불필요한 JOIN을 유발시킴으로써 코딩양을 증가시키고 검색 성능을 저하시키는 원인을 제공한다. NoSQL은 중첩 데이터 구조를 설계 할 수 있기 때문에 불필요한 JOIN을 최소화 시킬 수 있다.
- 4) 관계형 DB는 Entity 간의 N:M 관계 구조를 설계할 수 없지만 NoSQL은 N:M 관계 구조를 설계할 수 있고 구축할 수 있다.
- 5) Document DB는 기본적으로 Schema가 없기 때문에 유연한 데이터 구조를 설계할 수 있다.

MongoDB Design 기준

1. 데이터 조작은 어떻게 수행하는가 ?

- 1) **Dynamic Query** 사용 여부
- 2) **Secondary Indexes**의 사용여부
- 3) 원자적 **Update**의 실행 빈도
- 4) **Map Reduce**의 적용 여부

2. Access Patterns은 어떤가 ?

- 1) **Read**와 **Write**의 비율은 ?
- 2) **Update Type**은 어떤가 ?
- 3) **Data**의 **Life Cycle**은 어떤가 ?

3. 스키마 설계 시 고려 사항은 ?

- 1) **No Join**을 지향하는 데이터 구조 설계
- 2) **Linking & Embedded** 조건은 ?
- 3) **Document** 저장은 원자적 요소 기준으로 설계

주문전표

주문번호	2012-09-012345	담당사원	Magee		
고객명	Womansport (주)				
주문날짜	2012-09-20	선적날짜	2012-09-20	선적여부	Y
주문 총금액	601,100	지불방법	현금 30일 이내		

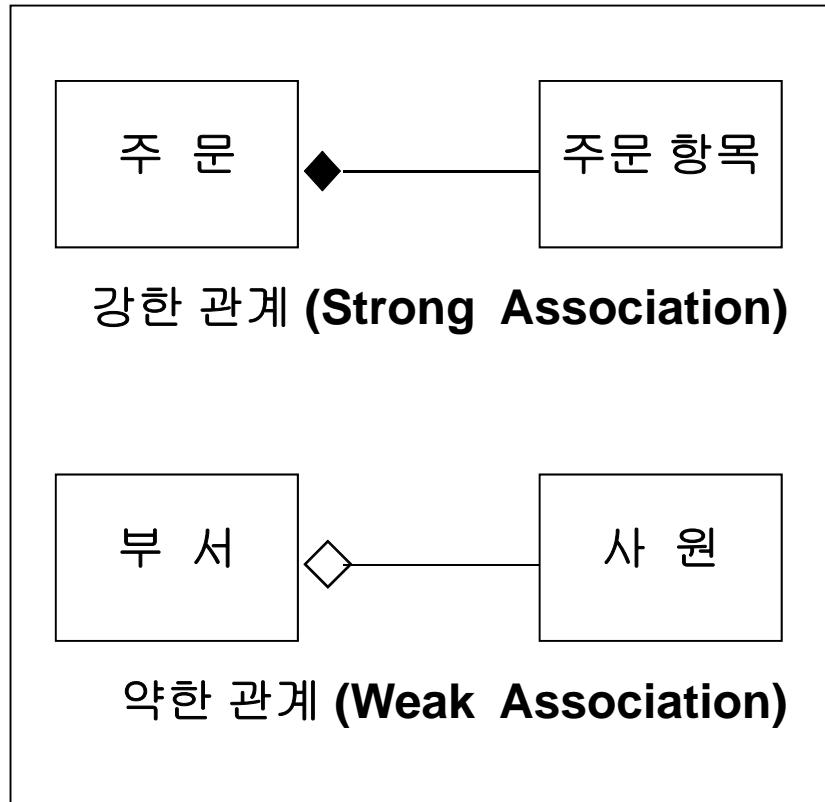
항목번호	제 품 명	단 가	주문수량	금 액
1	Bunny Boot	135	500	67,500
2	Pro Ski Boot	380	400	152,000
3	Bunny Ski Pole	14	500	7,000
4	Pro Ski Pole	36	400	14,400
5	Himalaya Bicycle	582	600	349,200
6	New Air Pump	20	450	9,000
7	Prostar 10Pd.Weight	8	250	2,000

SUMMIT2 (주)

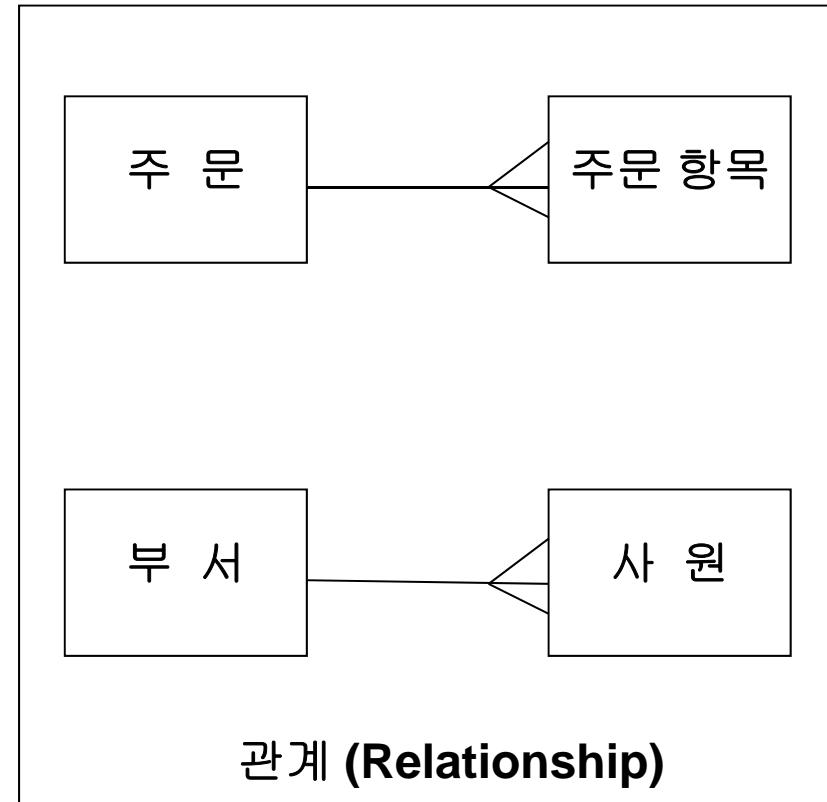
Schema 설계의 주요 특징

1) Embedded(Nested) & Linking 구조

- . 객체 지향 Data 관계유형과 관계형 Data 유형 모두를 설계할 수 있다.



Object Oriented Database



Relationship Database

MongoDB 데이터 저장 (Embedded)

```
db.ord.insert(
```

```
{ ord_id : "2012-09-012345",  
  customer_name : "Wonman & Sports",  
  emp_name : "Magee",  
  total : "601100",  
  payment_type : "Credit",  
  order_filled : "Y",
```

주문 공통 정보

```
  item_id : [ { item_id : "1",  
    product_name : "Bunny Boots",  
    item_price : "135",  
    qty : "500",  
    price : "67000" },  
    { item_id : "2",  
    product_name : "Pro Ski Boots",  
    item_price : "380",  
    qty : "400",  
    price : "152000" } ]
```

주문 상세 정보

```
})
```

MongoDB 데이터 저장 (Manual Linking)

```
> db.ord.insert( { ord_id : "2012-09-012345",  
customer_name : "Wonman & Sports",  
emp_name : "Magee",  
total : "601100",  
payment_type : "Credit",  
order_filled : "Y" } )
```

주문 공통 정보

```
> o = db.ord.findone( { "ord_id" : "2012-09-012345" } )  
{ "_id" : ObjectId("4fc21223e6cd4d2aadb38622"), .....
```

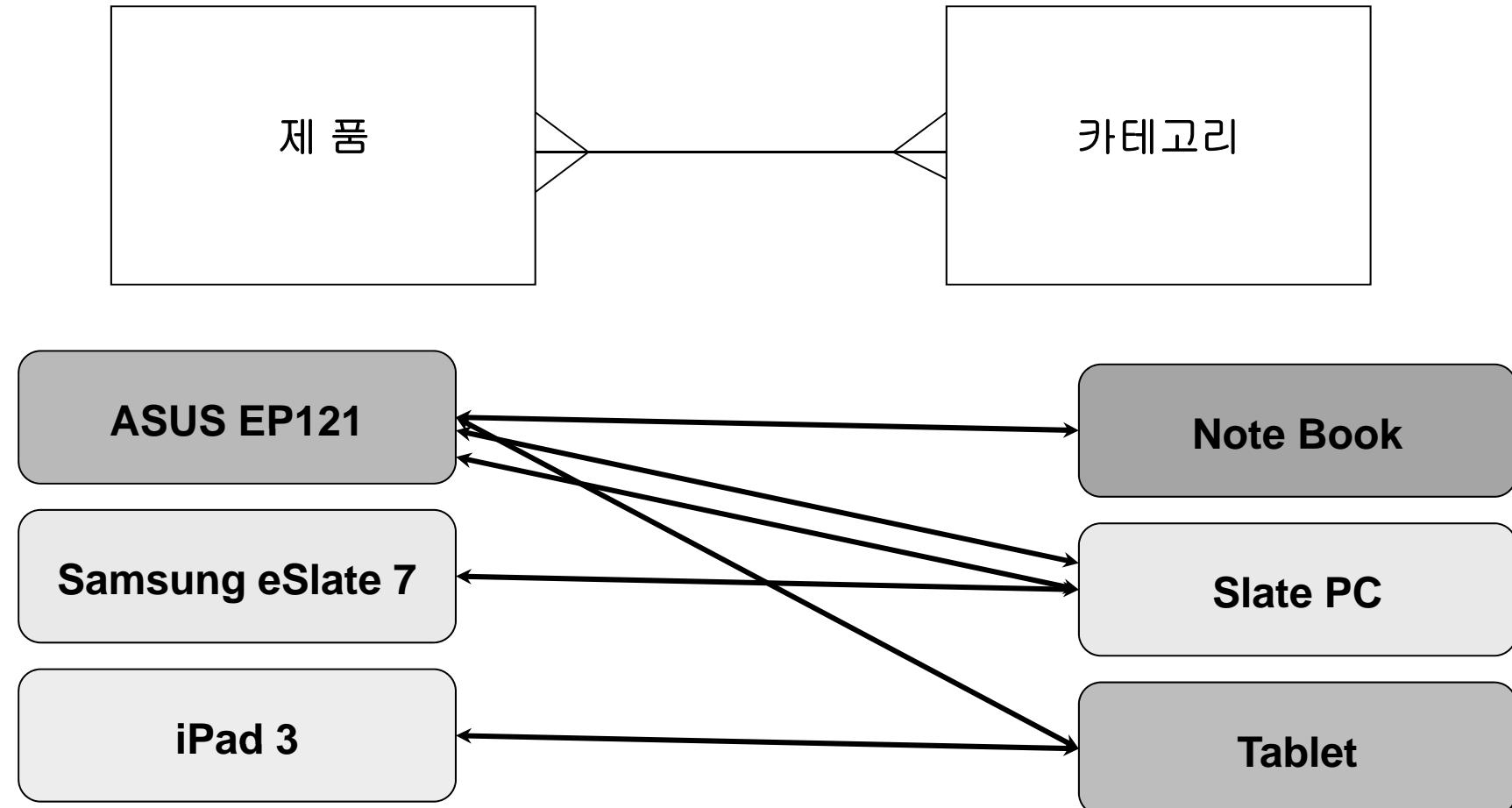
```
> db.ord_detail.insert( { ord_id : "2012-09-012345",  
item_id : [ { item_id : "1",  
product_name : "Bunny Boots",  
item_price : "135",  
qty : "500",  
price : "67000" },  
{ item_id : "2",  
product_name : "Pro Ski Boots",  
item_price : "380",  
qty : "400",  
price : "152000" }  
 ],  
ordid_id : ObjectId("4fc21223e6cd4d2aadb38622" ) } )
```

주문 상세 정보

```
> db.ord_detail.findOne({ordid_id : o._id})
```

2) Many To Many 관계 구조

- . 관계형 Data 구조에서는 제공하지 않은 다-대-다 관계 구조를 생성할 수 있다.

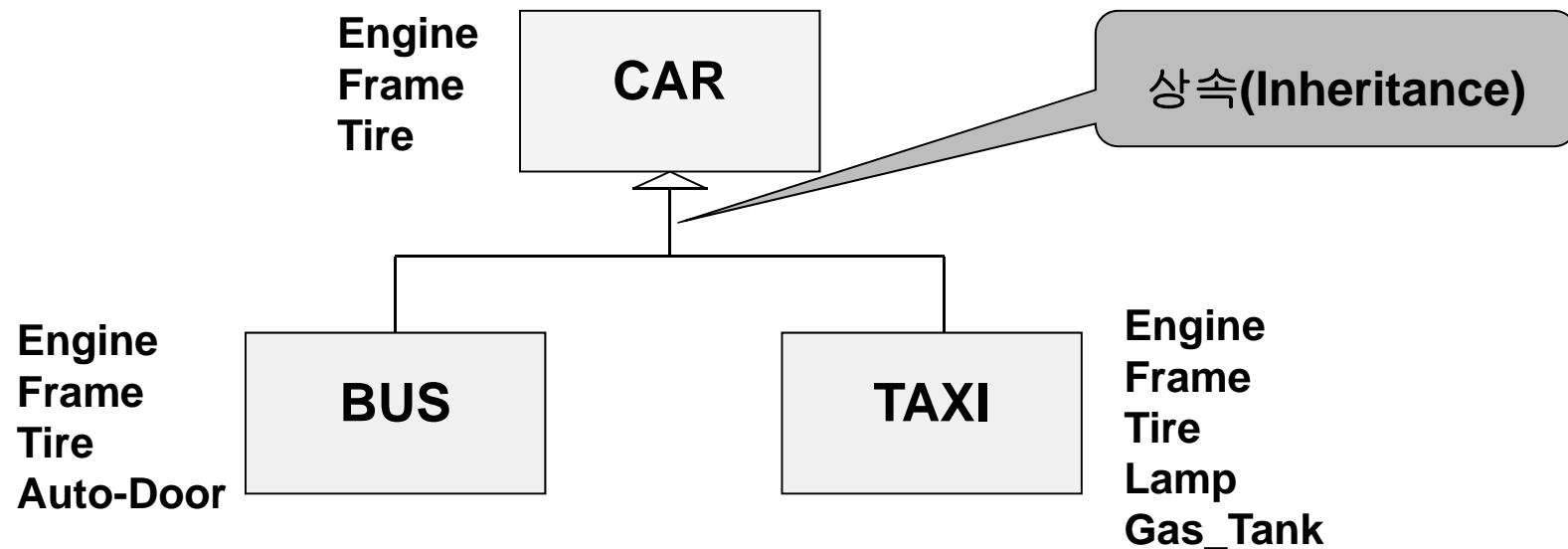


N:M 관계 (MongoDB)

```
> db.category.find()
{ "_id" : ObjectId("4fc23ac83153207db00dfd77"),
  "cname" : "Note Book", "pname1" : "Asus EP121 M50" }
{ "_id" : ObjectId("4fc23aca3153207db00dfd78"),
  "cname" : "Tablet",      "pname1" : "Asus EP121 M50",
                           "pname2" : "iPad3" }
{ "_id" : ObjectId("4fc23aca3153207db00dfd79"),
  "cname" : "SlatePC",    "pname1" : "Asus EP121 M50",
                           "pname2" : "Samsung eSlate 7" }

> db.product.find()
{ "_id" : ObjectId("4fc23adc3153207db00dfd7a"),
  "pname" : "Asus EP121 M50",
  "cname1" : "Note Book", "cname2" : "Tablet", "cname3" : "SlatePC" }
{ "_id" : ObjectId("4fc23adc3153207db00dfd7b"),
  "pname" : "Samsung eSlate 7", "cname1" : "SlatePC", }
{ "_id" : ObjectId("4fc23adc3153207db00dfd7c"),
  "pname" : "iPad3", "cname1" : "Tablet"}
```

3) Inheritance (OODBMS)



```
CREATE TYPE car AS OBJECT
(engine      NUMBER(9)    Primary Key,
 frame       VARCHAR(30),
 tire        VARCHAR(30)) NOT FINAL;
```

```
CREATE TYPE bus UNDER car_typ
(auto_door   VARCHAR(30) FINAL;
```

```
CREATE TYPE taxi UNDER car_typ
(lamp        VARCHAR(30),
 gas_tank   VARCHAR(30) FINAL;
```

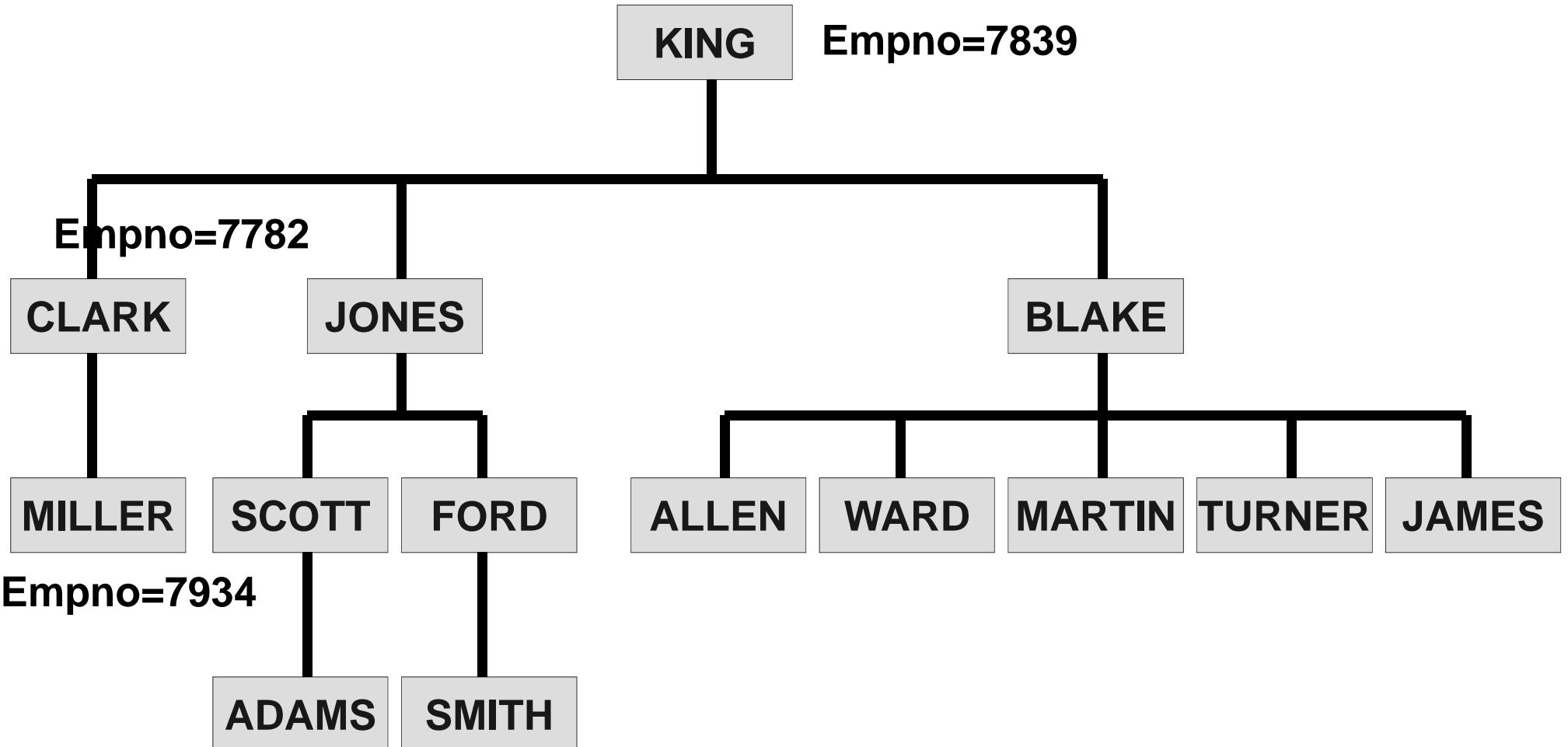
Single Table Inheritance (MongoDB)

```
> db.createCollection ("car");
> db.car.insert({ engine : "A", frame : "AX_1", tire : "R16",
                  car_type : "TAXI", lamp : 1, gas_tank : 1 });
> db.car.insert({ engine : "B", frame : "AK_3", tire : "R18",
                  car_type : "BUS", auto_door: 2 });
> db.car.insert({ engine : "A", frame : "AX_2", tire : "R18",
                  car_type : "TAXI", lamp : 2, gas_tank : 2 });

> db.car.find();
{"_id" : ObjectId("4f00574f81a153d6857897d2"),
 "engine" : "A", "ename" : "AX_1", "tire" : "R16",
 "car_type" : "TAXI", "lamp" : 1, "gas_tank" : 1}); }
```

Engine: A	Frame: AX_1	Tire: R16	Car_type: TAXI	Lamp: 1	Gas_tank: 1
Engine: B	Frame: AK_3	Tire: R18	Car_type: BUS	Auto_door: 1	
Engine: A	Frame: AX_1	Tire: R18	Car_type: TAXI	Lamp: 2	Gas_tank: 2

4) 계층형 데이터 구조



Self Reference Join (RDBMS)

Empno	ename	mgr	b.Ename
7839	KING		
7698	BLAKE	7839	KING
7782	CLARK	7839	KING
7566	JONES	7839	KING
7654	MARTIN	7698	BLAKE
7902	FORD	7566	JONES
7876	ADAMS	7788	JIMMY
7934	MILLER	7782	CLARK



```
SELECT a.empno, a.ename, a.mgr, b.ename  
FROM emp a, emp b  
WHERE a.mgr = b.empno
```

Ancestor Reference (MongoDB)

7839 : KING

7782 : CLARK

7934 : MILLER

```
> db.emp.insert({ "_id" : "7939", "name" : "KING",    "job" : "PRESIDENT" })
> db.emp.insert({ "_id" : "7782", "name" : "CLARK", "job" : "ANALYSIST",
                  "PARENT"      : "7839" } )
> db.emp.insert({ "_id" : "7934", "name" : "MILLER", "job" : "CLERK",
                  "ANCESTORS"   : [ "7939", "7782" ],
                  "PARENT"       : "7782" } )

> db.emp.find({"ANCESTORS" : "7939"})
{ "_id" : "7934", "name" : "MILLER", "job" : "CLERK",
  "ANCESTORS" : [ "7939", "7782" ], "PARENT" : "7782" }

> db.emp.find({"PARENT" : "7839"})
{ "_id" : "7782", "name" : "CLARK", "job" : "ANALYSIST",
  "PARENT" : "7839" }
```

- 1) 기업에서 발생하는 데이터 구조 중에는 계층형 데이터 구조가 발생할 수 밖에 없는데 이런 경우 적용하면 가장 이상적인 데이터 모델이다.
- 2) **ANCESTORS**와 **PARENT Field**로 표현할 수 있으며 하나의 **Document**는 하나 이상의 **ANCESTORS**와 **PARENT**를 가질 필요는 없다.